From feature selection to neural architecture search: Development and implementation of AI-based algorithms to enhance AI-driven research

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktor-Ingenieurin (Dr.-Ing.)

genehmigte Dissertation (kumulativ)

von
Elisabeth Johanna Schiessler

aus
Wien, Österreich

2024

Vorsitz des Prüfungsausschusses:	Prof. Dr.-Ing. Gerhard Bauch (Technische Universität Hamburg)
Gutachten:	Prof. Dr. Roland C. Aydin (Technische Universität Hamburg)
	Prof. Dr. Olaf Landsiedel (Christian Albrechts Universität zu Kiel)
Tag der mündlichen Prüfung:	19.09.2024

DOI: https://doi.org/10.15480/882.13706
ORCiD: https://orcid.org/0000-0001-5520-8325

Abstract

Effectively applying machine learning methods, particularly in applied sciences, can pose significant challenges. However, when employed correctly, these algorithms prove to be powerful tools, offering substantial benefits across a wide range of research applications. Fine-tuning them to individual needs and circumstances requires making a number of relevant and well-informed choices, all of which can profoundly impact the quality of the outcome. In this thesis, I present a comprehensive overview over the machine learning process, along with two use-cases of successful machine learning application in practice.

Kurzfassung

Maschinelles Lernen effektiv einzusetzen kann signifikante Herausforderungen mit sich bringen, insbesondere in den angewandten Wissenschaften. Wenn sie allerdings korrekt angewandt werden, so sind diese Algorithmen leistungsstarke Werkzeuge, die für eine Vielzahl von Forschungsanwendungen erheblichen Nutzen bringen können. Ihre Feinabstimmung auf individuelle Bedürfnisse und Umstände erfordert einer Reihe relevanter und fundierter Entscheidungen, die alle die Qualität der Ergebnisse maßgeblich beeinflussen können. In dieser Dissertation präsentiere ich einen umfassenden Überblick über den Prozess des maschinellen Lernens, sowie zwei Anwendungsbeispiele für erfolgreiche Anwendung von maschinellem Lernen in der Praxis.

Acknowledgements

Completing a PhD is no small feat, and takes a lot of time, energy, and commitment. I could not have achieved it without the continuing support and encouragement from a lot of people, and I am grateful and indebted to all of them.

First and foremost, I want to express my gratitude towards my thesis advisor, Prof. Dr. Roland Aydin, for his guidance and support throughout my four years at the Helmholtz-Zentrum Hereon. Thank you for your unwavering encouragement, for allowing me the freedom to work independently, and for your great sense of humour.

Next, I want to thank Prof. Dr. Christian Cyron, who served as my official thesis advisor for the majority of my time at Hereon. Thank you for your quick response times, detailed and constructive feedback, and abundant praise. I would also like to thank Prof. Dr. Olaf Landsiedel from Kiel University for allowing and coaching me to supervise my own master's thesis. Thank you for staying calm and relaxed when I became frustrated and stressed with things not always going the way I wanted them to.

My life at Hereon was greatly improved my a number of awesome people. I was able to join up with Christian Feiler and Tim Würger for two highly successful cooperations. Thank you both for becoming my surrogate group when working from home continued on and on forever, for bringing material sciences and me closer together, and for making group work a fun experience. Thank you to my fellow PhD colleagues Anju Chandran and Vasu Kolli for lots of lunchtime company, for regularly checking in on me, and for patiently learning a new board game whenever we meet up outside work. Thank you to Elisa Passos for being my first friend in an unknown environment, and for dragging me out on great adventures when all I wanted to do was spend too much time on my work. Thank you also to Stephanie Kropf-Eilers from Hereon and Bettina Schrieber from TUHH for all your administrative efforts, and for caring and taking good care of the well-being of your students.

My family and friends also played a huge part in carrying me through this thesis. A big round of thanks goes to Julia Dujmovits, Nora Muck, Hannah Wechsler, Iris Vukovics, the Mittagessenrunde, and the TU+ people. Thank you to my parents, Locki and Georg Schiessler, for always believing in me, and for cheering on each and every achievement. Thank you to my brilliant sister, Sabine Schiessler, for knowing what to say when I need to rant, for understanding me without explanations, and for just getting me. Thank you to Ariane Böhm and Thomas Seidl, for bringing calmness and rationality into my life, structure into my work, and, on occasion, sharing a bit of madness and insanity. A big thank you to Eva Grundschober and Richard Draxelmayr, for being my home whenever I have the opportunity to visit, for always making time when I need you, and for endlessly mobbing me so I don't forget how much I am loved.

And finally, a huge thank you goes to my partner, Elisabeth Graeber, who shares much more with me than just a name. You are my safe haven and my adventure; you ground me and grow with me, and you see me like no one else does. I am so glad to have you in my life.

Contents

Chapter 1

Introduction

We live in a dataful world

Today's world is all about data. In scientific research, the primary objective has always been to gather, organise, and understand information, and subsequently draw educated conclusions from one's findings. With the advent of widespread digitisation and an increased availability of affordable storage space, exhaustive data collection has extended well beyond the confines of academia.

These vast stores of information may be used to monitor operations, make various predictions, enhance sales and profits, or, quite commonly, not at all. A significant reason why companies or individuals often refrain from making use of the data they have gathered is that their sheer quantity and complexity frequently surpass what humans can effectively process and comprehend.

While computers facilitate statistical analyses, users may still struggle to evaluate the quality of the compiled information, or to discern intricate interconnections and complex patterns hidden within. It is feasible, albeit usually time-consuming and resource-intensive, to handcraft algorithms tailored to specific data processing and comprehension needs. In recent decades, there has thus been a growing interest in enabling computers to generate their own algorithms by simply studying enough examples and learning from the gathered material. This problem-solving approach, known as machine learning, is widely employed today.

Driven by the increasing popularity and accessibility of machine learning, leveraging amassed stores of data is becoming more viable. Typical applications encompass image recognition, comprehending text and mimicking language, predicting future events based on current or past developments, recommending content to users based on their previous choices, as well as various forms of exploratory data analyses. Within scientific communities, machine learning and particularly its subfield, deep learning, has been used for decades to achieve groundbreaking results, even instigating several highly specialised research fields.

1.1 Public visibility of artificial intelligence

Artificial intelligence, or AI, is a popular umbrella term encompassing machine learning, but often specifically referring to deep learning. In media representations, 'using AI' is often depicted as starting up a program on one's computer, feeding in some data, and with the click of a button, the machine immediately knows all the answers.* This notion of an omniscient program is referred to as *artificial general intelligence* (AGI), and is deemed unattainable with our current technological capabilities. [1]

General public awareness of AI has witnessed a notable increase in recent years. This surge coincides with the broader availability of tools and applications relying on such techniques. A multitude of platforms such as online marketplaces, streaming services for audio and video content, as well as social media, have long since integrated recommendation algorithms that provide suggestions based on interactions learned from previous users. (Semi-)autonomously driven vehicles have gained widespread popularity over the past 10 to 15 years, shedding light on computer vision challenges.

In early 2023, there was a significant spike in interest regarding large language models, a subset of deep learning, following the release of the tech company OpenAI's chatbot, ChatGPT, which was made accessible to the public free of charge. The latest version of its underlying language processing algorithm, GPT-4 (short for Generative Pre-trained Transformer-4), demonstrates a remarkable level of sophistication in comprehending questions and generating diverse responses. Consequently, this has ignited serious discussions about whether the pace of development in machine learning and artificial intelligence is exceeding our current society's capacity to handle it. A group of scientists and experts in the tech field even released an open letter, advocating for a pause in AI research. [2]

OpenAI achieved the development of such a capable chatbot by creating an exceedingly large model, even by modern standards. It was trained on vast amounts of text data using substantial computational resources, advanced infrastructure, and a multitude of highly qualified personnel. Although precise figures are not publicly available for the latest versions, GPT-4 and its predecessor GPT-3.5 both are built upon GPT-3, which boasts approximately 175 billion model parameters and was trained on roughly 570 GB of training data. [3] It is reasonable to assume that subsequent versions will maintain a similar scale and likely employ even larger training datasets.

In a small experiment, I queried ChatGPT (using the publicly available GPT-3.5 version) to translate '570 GB of text' into a more easily understandable representation, such as how much shelf space this text would fill if printed into average-sized books, and how long it would take to read all these books. The response estimated slightly over 30 km of shelf space (with spines visible, not covers) and approximately 100 years of reading time at an average pace. Notably, when asking the same question multiple times, the values varied considerably, sometimes by a factor of up to 10. This variability can be attributed to the chatbot occasionally misinterpreting specific phrasing or choosing different estimates for various calculation steps. Even in this small

*Which are all 42, obviously.

experiment, it is evident that AI-generated results must still be approached cautiously and interpreted with a grain of salt. The experiment was not repeated using the state-of-the-art GPT-4 version.

1.2 Machine learning in applied sciences

In general machine learning research, and even more so when making use of machine learning algorithms in applied sciences, results of similar magnitude and capabilities to those of ChatGPT are as of yet rare bordering on non-existent. This scarcity can be attributed to various factors, including the unavailability of computational resources, a lack of suitable training data in terms of both quality and quantity, and, not least, a shortage of proficient researchers. At first glance, the need for specialised user expertise may seem counterintuitive, given that the primary object of using machine learning methods was to enable computers to handle most of the workload. Regrettably, reality is not as straightforward.

Successfully applying machine learning in practice necessitates a well-thought-out preprocessing and training process, accompanied by a careful validation of one's results. The development of a machine learning pipeline involves several distinct steps. They range from data preprocessing, over hyperparameter and methodology choices, to a careful examination of the final outcome. Each of these steps presents its own set of challenges and pitfalls, and for each one, sufficient user expertise is required to guarantee results of high quality.

Considering the substantial demands and complexities, the value of employing machine learning methods might be questioned, particularly in fields with limited data availability. One such instance can be found in material sciences, specifically in the modelling and understanding of corrosion properties of magnesium.

This light metal element occurs quite abundantly on Earth[4] and exhibits a relatively high electrochemical reactivity[5]. Along with its alloys, it has the potential to become a key resource in several application fields, all of which require specific corrosion behaviour from magnesium-based components. In the automotive and aerospace industries, corrosion must be entirely prevented for structural elements.[6–8] As an anode material in battery applications, magnesium should degrade at a constant rate.[9,10] As a non-toxic, biodegradable implant, magnesium-based parts need to disintegrate at a rate suitable to the specific patient and injury.[11,12]

Control of this corrosion behaviour is achievable by introducing small organic compounds to either the metal or as a component of a coating system.[13–15] The chemical space from which such compounds can be selected is almost unlimited[16], rendering a purely experimental approach to finding suitable candidates unfeasible. Moreover, it remains unclear which properties of these compounds are most relevant concerning their efficacy in corrosion control. As a result, data-driven approaches[17–19] and computational techniques[20–22] are essential for understanding and predicting the ways in which corrosion behaviour is linked to material characteristics of these compounds. This search is further complicated by the fact that available datasets on magnesium corrosion inhibitors[14] are quite diminutive in terms of machine learning.

Machine learning algorithms prove to be particularly powerful tools for discerning the often intricate and non-linear interdependencies hidden within such a vast and complex environment, rendering them ideal candidates for the task of identifying the most relevant material characteristics. Once a suitable set of such features has been determined, a predictive model can be trained on them, with the hope of providing insights into the corrosion control behaviour of materials that have not been previously tested.

1.3 Researching deep learning methods

As the complexity of tasks increases, so do the demands on the sophistication and flexibility of algorithms. Deep learning is a subfield of machine learning that uses neural networks, which are currently the most powerful class of machine learning algorithms available. These networks are modelled to resemble the structure and learning abilities of the human brain. When an application is based on or 'uses AI', it is likely that deep learning is the driving force behind it.

Neural networks can be customised in numerous ways, including adjusting the number and type of layers, hyperparameter settings within the layers, and specifics of the training process. They particularly excel when there is a substantial amount of training data available. Although there is no clearly defined threshold value for when to use deep learning, as a general guideline, once a dataset comprises more than a few hundred samples, deep learning becomes a viable option. As we will demonstrate later, deep learning can also be effectively applied to much smaller datasets if appropriate preprocessing and validation steps are followed.

For developing and refining deep learning methods, researchers typically rely on large, well-known benchmark datasets, such as ImageNet[23], a database containing over 14 million hand-labelled images. Annual software competitions for image classification, held from 2010 to 2017, led to the creation of famous network architectures such as AlexNet[24] or ResNet[25]. Many scientific fields that depend on computer vision tasks, such as autonomous driving[26], or medical imaging for cancer detection[27,28], have directly benefited from these advancements.

The process of training a neural network can be highly resource-intensive in terms of both time and computational power. Consequently, selecting an appropriate network topology and choosing sensible training hyperparameters is of utmost importance to avoid unnecessary expenses. Making sensible and effective choices often requires considerable user expertise. The sub-discipline of deep learning which focuses on automating such choices as much as possible is known as neural architecture search.[29,30]

Neural architecture search algorithms typically have access to a range of network topologies (or architectures) from which they can choose. They select one or more of these candidate architectures, train them, and then evaluate their performance. The evaluation results are used to update the search algorithm's rules and preferences. This process is continued iteratively, until a final network architecture is established.[29,31] While employing a neural architecture search algorithm is expected to be more efficient and yield superior results compared to manual configuration testing, it can still be quite resource-intensive.[30] Some of these algorithms use

meta-controllers, separate neural networks responsible for choosing candidate topologies and learning associated rules and preferences. These meta-controllers may require pre-training on task specifics before they can be employed.

Many neural architecture search algorithms tend to create rather large, overparameterised networks[30], which, particularly in cases with limited training data, can be detrimental to the predictive quality and generalisation ability of the final neural network. Hence, it is crucial to develop algorithms that not only produce large, powerful topologies trained on extensive datasets like ImageNet, capable of achieving state-of-the-art accuracy levels, but also smaller, more lightweight models focused on user-friendliness and computational efficiency.

1.4 Scope of this thesis

In this thesis, I aim to address the following objectives:

1. Provide a comprehensive guide for understanding the machine learning process. This thesis is intended as a handbook that explains the importance, theory, and background of each step, along with two use-cases that showcase practical applications of the presented tools.

2. Identify relevant material properties related to magnesium corrosion control. Using a database of 60 small organic compounds, feature selection methods are employed to determine which of their more than 1,000 describing attributes are most relevant in predicting a compound's potential for corrosion inhibition. The selected features are then used as inputs for training predictive neural networks. The generalisability of the sparse predictive models is validated by applying them to a new set of previously unseen samples. The attributes established earlier are evaluated using this extended dataset, and chemical structures lying outside the predictive domain of the model are identified.

3. Develop a user-friendly, cost-effective neural architecture search algorithm. It can autonomously select the most appropriate neural network architecture and topological elements for a given dataset and research question. By making small modifications to existing network architectures that retain their overall input-output behaviour, candidate architectures are identified that can overcome local optima and thus have increased predictive capabilities. The algorithm is simple and easy to use, and incorporates topology optimisation into the training process. It is capable of running on limited data, significantly increasing the predictive capabilities of a given network topology, or even 'rescuing' architectures that, on their own, are insufficient for the complexity of a given task and dataset. This enables even inexperienced users to successfully employ deep learning techniques in their research.

1.5 Outline of this thesis

The remainder of this work is structured as follows:

- Chapter 2 gives a step-by-step overview of practical considerations when applying machine learning methods for predictive modelling. It explains the rationale for dividing available datasets into different subsets and the benefits of a structured validation process. Furthermore, it introduces the topics of feature selection and feature engineering, along with strategies for automatically selecting appropriate steps for the machine learning pipeline.
- Chapter 3 introduces common types and structural elements of neural networks, along with a notation for their mathematical representation. It also explains how neural networks are trained.
- Chapter 4 presents tools and methods for automating neural network-based machine learning. It covers topics such as neural architecture search, pruning techniques, and the concept of genetic algorithms.
- Chapter 5 presents two use-cases of AI-driven research in practice. It explains how to incorporate the tools and methods presented in the previous chapters. Results and achievements from several publications are highlighted and summarised.
- Chapter 6 provides an overall summary and discussion, and gives a general outlook.

The peer reviewed and published manuscripts upon which this thesis is based are included in Appendix A. Another manuscript in preparation for submission, which is also incorporated into this thesis, is included in Appendix B.

Chapter 2

Practical concerns in applied machine learning

Building the pipeline

Successfully employing machine learning in practice is an intricate process that begins with determining which questions one wishes to address. In terms of datasets, the answers to these questions are contained in a designated target variable. While some use-cases of machine learning aim to predict multiple target variables simultaneously, this leads to a more challenging class of problems that fall outside the scope of this thesis. Several additional preprocessing and selection steps are required before a machine learning algorithm can be trained to make predictions on previously unseen data. Finally, a meticulous validation process is crucial for assessing the generalisability of the model before placing any amount of confidence in predictive results.

This chapter highlights the most important aspects of inspecting and preprocessing datasets for machine learning. It explains the rationale behind splitting available data into separate sets, and why using all that is known for training a machine learning algorithm is inadvisable. It sheds some light on why having more data available is not always better, and how to identify the parts containing relevant information. Furthermore, this chapter provides an overview over a number of tools and techniques that can be used to automate the machine learning process and aid in selecting relevant methods and hyperparameter settings for any given dataset and task.

2.1 Data preparation

The task of data preparation and cleaning is often referred to as the most crucial step in machine learning,[32] and neglecting it can have severe consequences. In the 'best' case scenario, working with messy data can cause algorithms or even the entire training process to fail. Unfortunately, as a common consequence of neglecting data preparation, one might end up with seemingly

plausible results that hide errors or are entirely incorrect. In computer science communities, this very common problem is often colloquially referred to as 'Garbage in, Garbage out'. This memorable phrase emphasises the importance of appropriately handling data before feeding them into any algorithm, as failure to do so is highly likely to yield subpar results.

Therefore, a closer examination of the available training data is necessary. Structural aspects, such as whether one is working with tabular or spatial data (e.g., images where the relative position of each pixel is relevant), become apparent relatively quickly. Target variables can take different forms: categorical data is used in classification tasks, like determining if a mass of cells on an MRI (magnetic resonance imaging) scan is malignant, or benign.[33] Numerical data is handled in regression tasks, such as predicting the market price of a house.[34] Ordinal data, such as predicting the category of a hurricane[35], can be a bit more challenging and is often treated as either a classification or regression task depending on the information available within the training data. Different types of algorithms are required to address these diverse types of problems.

The quantity of available data is a major concern for machine learning. Not all algorithms are suitable for handling all dataset sizes. A more complex one may tend to overfit, and struggle to generalise if the training set is too small. This means that the algorithm has learned the precise characteristics of the training data too well, and can provide exact answers for examples it has seen before. In such cases however, extrapolation on new samples usually yields poor results because the algorithm has generated overly strict rules, and as a consequence is not able to handle attribute values (or combinations thereof) that fall outside of narrow ranges.

On very large datasets, analysing all samples and deriving appropriate rules may be too challenging for simpler models, leading to excessive processing times or even out-of-memory issues. Complex deep learning architectures with numerous trainable parameters are particularly susceptible to memory overflow, especially when the training data is loaded into memory all at once.

The size of a dataset is measured not only in the number of samples, but also the number of attributes (also referred to as features) that describe each sample. In the case of spatial data (such as images), the term features is synonymous to all the individual pixels that comprise the image. The more features there are, the more complex the rules that machine learning algorithms need to learn, the more samples are required. However, not all features carry the same amount of information, and some (or many) may be superfluous; keeping them in the training set can even be detrimental to algorithm performance.[36] Identifying the relevant ones is a distinct machine learning problem known as as feature selection, and is covered in detail in Section 2.2.

Another major concern is the quality of the available data. Missing values, duplicates, or outliers can pose challenges for many machine learning algorithms. Furthermore, features that describe a dataset can come in different, often mixed, types, which not every algorithm can handle. Even if all features are purely numerical, they may contain variables from vastly different ranges which again can cause unwanted behaviour or lead to biased results. Methods that handle missing values, features of non-numeric types, as well as scaling belong to the class

of feature engineering algorithms. They are covered in Section 2.3.

Fortunately, a variety of tools are available to assist in ensuring both sufficient data quality[37] and quantity[38]. What can not be automated however is the process of properly understanding what the data look like, how they are structured, and whether or not they are even suitable for answering the intended research question.

2.2 Feature selection

As previously mentioned, the size of a dataset is determined both by the number of samples contained within it, as well as the number of features or attributes describing each sample. Generally speaking, the more samples are available for training a machine learning algorithm, the better. However, describing data by more features is not always beneficial.[36]

With a rising number of attributes, the distribution of existing samples becomes more and more sparse. This is due to the fact that the volume of (mathematical) spaces increases exponentially with each added dimension. In particular, the number of samples required to counteract this increasing sparsity grows exponentially with the number of features. This problem is known as 'the curse of dimensionality'[39], and is illustrated in Figure 2.1.

Usually, the majority of the information content required to successfully train a machine learning algorithm is contained in only a handful of features; the rest are superfluous noise or redundant, and including them in the training process can even be detrimental to the quality of the output.[36]

While modern computers are becoming more capable and can handle larger datasets without memory issues, computational resources are often still a limiting factor, especially if the fully trained model later needs to be deployed on a mobile device. Furthermore, the process of data gathering itself (e.g. collecting samples from patients, performing experiments in a laboratory, annotating images) can be quite costly and time-consuming.

All in all, finding out which attributes play the most important roles for predicting the target variable is a crucial step when using machine learning in practise. The field concerned with this task is called feature selection.

Feature selection algorithms can be grouped into three different types:[40,41]

1. Filter methods
2. Wrapper methods
3. Embedded methods

1. Filter methods rank features via some form of scoring criterion. This criterion can be correlation-based or make use of basic statistical measurements such as the χ^2- (Chi-squared-) statistics, information gain, or analysis of variance (ANOVA), and is calculated directly from

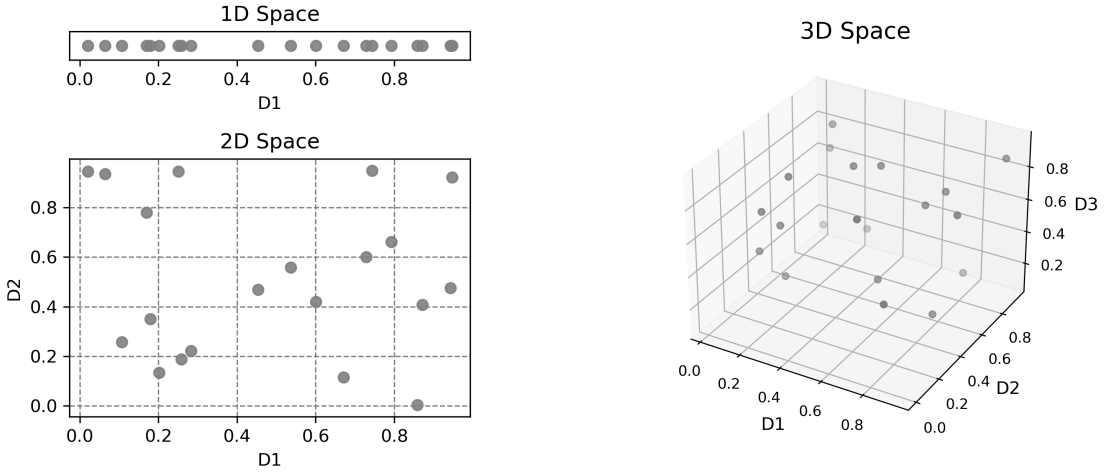


Figure 2.1: Plots of the first (top left), first and second (bottom left), and all three (right) dimensions of 20 randomly sampled three-dimensional data points. With rising number of dimensions, the sparsity of the data increases. Reproduced from Cunningham et al. [40].

the training data. Then, a selection strategy is applied to choose features from the ranked list. Examples of such strategies include taking the top $x\%$, the k top-ranked features, or all features with a score larger than a certain threshold. The predictive model is then trained on the selected subset.

2. Wrapper methods train the predictive model on multiple different subsets of all available features and then select the best-performing one. In other words, training the predictor is wrapped into the search process. Examples of wrappers include exhaustive search, were each and every available subset of features is tested. Another popular wrapper technique is sequential forward selection (SFS). This algorithm trains the predictor on each single feature, identifies the best-performing one, and subsequently tests all pairs of two features that include the previously selected best-performing one. This process is repeated iteratively until the desired number of features has been reached or no more improvement occurs. Backward elimination (BE) works using the same principle but removes features one by one rather than sequentially adding selected ones.

Wrapper methods are much more computationally costly than filter methods since the predictive model is trained multiple times. The SFS procedure mentioned above needs to compare at least $n + (n-1) + (n-2) + \ldots + (n-(k-1)) = k \cdot n - \frac{k \cdot (k-1)}{2}$ fully trained models, where n is the total number of features and k the number that is selected. However, wrapper methods are also much better suited than filters for capturing the interconnections and dependencies between features, as filter methods often miss these when regarding features as completely independent.

3. Embedded methods naturally encompass feature selection into the training process. These methods include decision trees (DT), random forests (RF), as well as logistic regression methods. In decision trees, the feature that carries the most importance regarding predicting the target variable is used for the first splitting point, and so forth. Decision trees often only use a subset of all available features for their decision points, leading to a naturally emerging ranking and selection.

Another popular feature selection method is called recursive feature elimination (RFE). It is similar to BE in the sense that unimportant features are iteratively discarded. However, in order to do so, it relies on feature rankings obtained from an underlying predictor such as a decision tree or random forest, which is retrained once in every iteration. RFE combines elements of both filter and wrapper methods and is usually counted as an embedded method.

A potential baseline against which the effectiveness of feature selection methods can be tested is random sampling. A (random or pre-determined) number of features is selected using a random number generator. To counter potential statistical artefacts, this selection method should be repeated several times and averages be taken over final predictive results. Another baseline for comparison should always be the complete set of all available features.

2.3 Feature engineering

Instead of selecting subsets of the available features in a dataset, it is sometimes useful to 'create' new features. This process is referred to as feature engineering. It encompasses pre-processing methods that handle missing values, as well as data from varying ranges or mixed types. Feature engineering is not only used as a preprocessing step to clean up the available data. Instead, attributes can also be re-combined or split up to condense information or make previously hidden parts accessible.

Samples with missing values are often omitted during training, as many algorithms are unable to handle such cases. This can significantly reduce the effective dataset size when many instances are affected, or introduce unwanted bias if, for example, only samples belonging to a certain class have missing values. Substitutes for these blanks can be created using a fixed default value, or various statistical measures, such as inserting the median or mean values. For time series data, the last known number is kept until the next populated sample is reached. This process is called data imputation.[42]

Numerical features often come in varying ranges. For instance, the melting points of most metals can range from several hundred to a few thousand degrees Celsius. Their densities, on the other hand, usually vary between around one to a few dozen grams per cubic centimetre at most.[43] If used unscaled, the melting point attribute would most likely be given much more importance by a machine learning algorithm, even if the target variable correlates only very loosely with it. Features are therefore usually either scaled or normalised such that their ranges or distributions are similar. A common feature scaling method is min-max scaling, where the range of each feature is determined and then transformed into e.g. the range $[0, 1]$ or $[-1, 1]$ via

linear transformation. Normalising feature ranges, also known as standardisation or standard scaling, means transforming the data such that they have a mean of 0 and a standard deviation of 1.

Non-numeric or mixed type data are not always suitable as inputs for a chosen machine learning algorithm. Therefore, categorical and ordinal data are usually encoded. This means turning categories such as 'cat', 'dog' and 'house plant' into numerical values 1, 2 and 3. With categorical data, there is no inherent order or relation between the numbers representing the different categories. If a sample of a house plant (category 3) were incorrectly marked as category 1, the error is not greater than if the value were 2. Ordinal data, on the other hand, represents ordered categories, hence the name. Measurements taken during a category 5 hurricane can be expected to differ more from those of a category 1 than a category 4 one.

A very common feature engineering method that can also reduce dataset size is *principal component analysis* (PCA).[44–46] This concept stems from multivariate statistics, where it is used to represent high-dimensional data via projection onto a (potentially much smaller) number of orthogonal variables. These principal components stem from the eigenvectors of the data's covariance matrix.

When dealing with time-dependent data, extracting time-related features such as day of the week, month, or season from timestamps is also considered to be a feature engineering technique.

2.4 Post-training feature evaluation

Thus far, all discussed feature selection and feature engineering methods were applied directly to the 'raw' training data. There also exist some more advanced techniques that can be viewed as complex, 'a-posteriori' feature selection or engineering. These emerge during neural network training or are derived from fully trained networks.

Such methods include autoencoders[47,48], a type of neural network that aims to generate a compressed, lower-dimensional representation of the inputs. These representations are somewhat similar to a non-linear version of PCA, making autoencoders a kind of feature engineering technique. This type of neural networks is explained in more detail in Section 3.4.2.

In the context of explainable AI, which is concerned with making the decision-making process of machine learning more transparent, a popular tool for estimating the importance of individual features of a neural network are SHAP values (SHapley Additive exPlanations).[49,50] This concept stems from cooperative game theory and aims to measure the 'worth' of each attribute for predicting the target value within coalitions with other attributes, making it a form of advanced feature importance ranking method. SHAP values can be calculated for any fully trained neural networks.

2.5 Algorithm and hyperparameter selection

For each step of the machine learning process, a variety of tools and algorithms are available. Unfortunately, none of them are one-size-fits-all solutions, as different methods have individual strengths and weaknesses.[51] Their effectiveness depends on the available data and the specific task at hand. Many algorithms have tuneable hyperparameters, further increasing the choices required when building a machine learning application.

The performance of a model critically depends on selecting appropriate algorithms and hyperparameters. This task can be time-consuming and computationally expensive, even for experienced users. The field that aims to automate this process is called AutoML.[52] This term encompasses both the search for the best-suited algorithm, as well as the optimisation of related algorithm hyperparameters. The choice of the specific predictive machine learning algorithm can also be considered a hyperparameter of the machine learning process. AutoML is often used synonymously with hyperparameter optimisation.

Hyperparameter optimisation algorithms come in varying levels of complexity and capability. Exhaustive search, grid search and random search methods explore some or all possible combinations of parameter values in a structured (or random) manner. Statistically driven techniques, such as Bayesian optimisation[53], rely on statistical criteria to iteratively update their search strategy based on the performance of evaluated samples. Localised search algorithms, such as hill climbers[54], compare neighbouring solutions and iteratively update their selection towards higher-scoring ones. Hill climbers may be coupled with stochastic methods, such as genetic algorithms[55] or simulated annealing[56], to generate neighbour candidates. These methods aim to balance exploiting promising regions of hyperparameter combinations versus exploring previously unseen ones. As ever, each of these tools has its individual strengths and weaknesses, and the expense of implementing and running them must be weighed against their advantages over other search algorithms.

2.6 Algorithm performance metrics

Depending on the dataset and task, different metrics are available to measure how well a machine learning algorithm performs. For classification tasks, the most common indicator for algorithm performance is called accuracy. This measure describes the fraction (or percentage) of samples that were labelled using the correct class label:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{2.1}$$

Other common metrics for classification tasks include precision, sensitivity, specificity, and composites thereof. These are most often stated for binary classification tasks, where the class values can be represented as ‘true’ or ‘false’, but can all be adapted for multi-class classification tasks. In such cases, they need to be calculated per class, where ‘true’ indicates membership of the regarded class and ‘false’ that of any other class.

Precision measures how many of the samples predicted as positive are truly members of the positive class:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives } + \text{ false positives}} \tag{2.2}$$

Sensitivity, also known as recall, measures how many samples belonging to the positive class were correctly identified as such:

$$\text{Sensitivity (Recall)} = \frac{\text{True positives}}{\text{True positives } + \text{ false negatives}} \tag{2.3}$$

Specificity, also known as selectivity, measures how many samples belonging to the negative class (or to any other but the regarded class in case of multi-class problems) were correctly identified as such:

$$\text{Specificity (Selectivity)} = \frac{\text{True negatives}}{\text{True negatives } + \text{ false positives}} \tag{2.4}$$

All four of the presented metrics output values between 0.0 and 1.0, with 1.0 indicating a well-performing algorithm, though not necessarily a perfect prediction of each sample. A precision score of 1.0 means that all samples predicted as positives are truly positive. False negatives are not taken into account by this metric, however. In particular, an algorithm that always outputs positive will achieve a precision score of 1.0, but could theoretically have an accuracy of 0.0 if all regarded samples belong to the negative class. Therefore, precision and recall are often reported together, usually in combination with the F1-score, which is the harmonic mean between them:

$$\text{F1} - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

A practical example of a binary classification task is virus testing, with test results belonging to the two classes 'positive' and 'negative'. Consider a group of 100 people, out of which 60 are infected with a virus, whereas 40 are not. A rapid test correctly identified 54 of the 60 infected people as positive ('true positives'), and 38 of the 40 not infected people as negative ('true negatives'). The remaining 8 were misdiagnosed by the test (2 are 'false positives' and 6 are 'false negatives').

This test therefore has an accuracy of $(54 + 38)/100 = 92/100 = 0.92$, having made correct predictions in 92% of all cases. The test's precision score is $54/(54 + 2) = 54/56 = 0.96$. This means that out of the 56 people testing positive, 96% truly are infected. Its sensitivity score is $54/(54 + 6) = 54/60 = 0.9$, meaning that 90% of infected people were correctly identified. And finally, the specificity score of the test is $38/(38 + 2) = 38/40 = 0.95$, indicating that 95% of individuals with a negative result are indeed not infected. In particular, while all three metrics may appear to be highly similar, they convey different aspects of the quality of the applied test.

Lastly, the virus testing example reaches an F1-score of 0.93. As with the other metrics, the higher the F1-score, the better. However, no universal scale or threshold exists that indicates

when a score can be considered 'good' (or 'good enough'). Such assessments can only be made in comparison to other methods, that may reach different scores.

2.6.1 Loss functions

Instead of measuring how well an algorithm is doing, it is also possible to look at how badly it performs. This can be achieved via a so-called loss function. Regression tasks, which output a numeric value instead of a category or class, are typically evaluated using only a loss function. Common measures for the error made by a regression model include the mean squared error (MSE) given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.6}$$

where n is the number of samples, y_i the predicted and $\hat{y}_i$ the true value for sample i. Sometimes, the root mean squared error (RMSE), given by

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{2.7}$$

is preferred over the MSE.

Both the MSE and the RMSE lack pre-determined ranges and should be regarded in relation to the range of the target value. Therefore, other statistical measures are sometimes reported along with MSE or RMSE values. Common amongst these is the coefficient of determination, or R^2 value. This metric evaluates the goodness of fit of a regression model by comparing the variances of the predicted values to the variances of the true values. The coefficient of determination is given by

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{2.8}$$

where $\bar{y}$ indicates the mean of the true values, and the other variables are used as above in Equation (2.6). R^2 normally ranges from -1.0 to 1.0, with 1.0 indicating perfect predictions, and negative values meaning that the model performs worse than if it would always predict the sample mean.

Classification tasks by definition do not produce numerical results, and can therefore not be evaluated using the above loss functions. However, some types of machine learning algorithms, such as neural networks which are discussed in detail in Chapter 3, rely on having a loss function available. Under the hood, these algorithms often learn probabilities of samples belonging to each of the available classes. Functions such as the binary or categorical cross-entropy loss then compare the differences between probability distributions.[57]

2.6.2 Dataset splitting and cross-validation

A very important step in the machine learning process is performing a train-test split. This involves dividing the available dataset into two parts, typically of uneven sizes. The larger

part, known as the training split or training set (denoted by $\mathcal{D}$), is used to train the machine learning algorithm.

The smaller part constitutes the test set, $\mathcal{D}_{\text{test}}$. Having a distinct test set enables the estimation of how well the machine learning algorithm can perform on entirely new, previously unseen data – or, in other words, its ability to generalise. Therefore metrics used to measure the predictive capabilities of a machine learning model should always be calculated on this split.

Without a separate test set, any evaluation metric merely gauges how well the chosen machine learning model captures the specific patterns and noise present within the training data. Algorithms that excel at capturing these structures are considered overfitted to the training data. They tend to perform well on the training set but poorly on the test set.

Especially in the case of small or unevenly distributed datasets, the selection of the train-test split can significantly impact model performance. A common technique to address such statistical artefacts is called cross-validation (CV). During CV, after setting aside a designated test set, the training set is once again divided into n equal parts, called folds. Subsequently, all preprocessing, feature or model selection, and training steps are repeated n times. Each time, a different part of of the training data is set aside as a validation set, $\mathcal{D}_{\text{valid}}$. Performance metrics are calculated on $\mathcal{D}_{\text{valid}}$ and then averaged over all folds. Finally, the performance on $\mathcal{D}_{\text{test}}$ also needs to be evaluated for all n versions and then averaged. Cross-validation enhances the generalisability of results.

Unfortunately, the terms test and validation set are applied inconsistently throughout the machine learning community and pertinent literature. It is therefore important to always pay close attention to their specific usage in any given text or context.

Chapter 3

Neural network-based machine learning

Mimicking the human brain

There is an inherent limit to the complexity of tasks and datasets that each different type of machine learning algorithm is able to handle. With rising amounts of available training data, and access to sufficient computational power, the demands on model capacity increase simultaneously. *Artificial neural networks* (ANNs), also abbreviated as neural networks, are the most powerful type of machine learning algorithm available today.[57] These networks are modelled after the neurons in a human brain, where information from multiple sources is collected via structures called dendrites, aggregated in the cell nucleus, processed and then passed on.[58] ANNs can link several layers of neurons, in which case they are referred to as *deep* networks. Neural network based machine learning algorithms are therefore frequently summarised as *deep learning* (DL). In addition to being able to handle larger and more complex datasets, these algorithms exhibit plasticity. This means that they are able to learn new information once it becomes available, without having to be re-trained entirely from scratch.[57]

The idea of neural networks stems from mathematical modelling; therefore the language best suited to represent them is inherently mathematical. This chapter introduces the mathematical notations required to describe and manipulate elements of ANNs. It presents the basics of neural networks and deep learning, and elaborates on how to train neural networks. Common (structural) elements of neural networks are introduced, and an outlook is given into more advanced types of networks.

3.1 Mathematical representations of neural networks

The most basic structure of a neural network is the *(single-layer) perceptron*[59], which models the combination of two or more input values into a single output value, often depicted using circles and lines as illustrated in Figure 3.1. In such depictions, information is always considered

to flow from left to right unless explicitly marked by arrows pointing out the direction. The circles are called *neurons* or *nodes*, and represent scalar valued information units. The terms neuron and node are often used interchangeably, with neuron stemming from the biological, and node from the graph theory point of view.

The neurons are connected via weights, marked by lines. These weights are given in the form of a weight vector $\boldsymbol{w}$. The respective input values are multiplied with their weights and then summed up. Sometimes a bias term b, a constant not influenced by the input values, is also added. Most depictions of perceptrons skip this element.

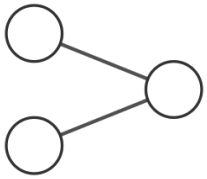


Figure 3.1: A single-layer perceptron. Two input nodes (left) are connected via weights to a single output node (right). Figure created using LeNail[60].

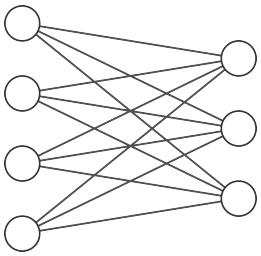


Figure 3.2: A multi-class perceptron, that transforms four dimensional inputs into three different classes. Figure created using LeNail[60].

Perceptrons are designed to distinguish between the binary states zero and one. In order to do so, they aggregate the weighted inputs, and then pass the computed value through an *activation function*. This function is typically a form of threshold function, yielding either zero or one depending on the calculated activation state. By definition, single-layer perceptrons are therefore also *single-class* perceptrons, meaning that they can decide whether or not the provided inputs belong to a single class (or equivalently, contain a single property, fulfil a single condition, etc.). In mathematical terms, the output of a perceptron results to

$$f(\boldsymbol{x}) = \Phi(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \begin{cases} 1, & \boldsymbol{w} \cdot \boldsymbol{x} + b > c \\ 0, & \text{otherwise} \end{cases}, \tag{3.1}$$

with input $\boldsymbol{x}$, weights $\boldsymbol{w}$, bias term b and threshold c (often chosen as zero). Φ denotes the activation function, which in this case is a binary decision function (also known as *Heaviside step function*).[57]

Multi-class perceptrons are a combination of several single-class perceptrons, and w.l.o.g. transform m-dimensional inputs into n-dimensional outputs, as depicted in Figure 3.2. In this case, the connecting weights are represented as weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$. The i^{th} column vector $\boldsymbol{w}_i$ of $\boldsymbol{W}$ denotes the weights connecting the i^{th} input neuron to each of the n output neurons. The activation function Φ can also take more generalised, often non-linear forms, which yields

$$f(\boldsymbol{x}) = \Phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{3.2}$$

as the governing equation describing the output of a multi-class perceptron.[57] Activation functions and their properties are covered in more detail in Section 3.3.3.

When two or more multi-class perceptrons are combined in sequence (sometimes described as 'stacked on top of each other', as shown in Figure 3.3), they form the layers of a *deep* or *hidden belief* network, or simply a *multi-layer* perceptron (MLP)[61]. The output $f_i(\boldsymbol{x}^i)$ of one layer becomes the input $\boldsymbol{x}^{i+1}$ of the subsequent layer. Layers in the middle are referred to as *hidden layers* since neither their inputs nor outputs are directly observed from the outside. The governing equation $F(\cdot)$ describing a deep feed-forward neural network with k layers is thus given by

$$F(\boldsymbol{x}) = f_k(\boldsymbol{x}^k) = f_k \circ f_{k-1}(\boldsymbol{x}^{k-1}) = \ldots = f_k \circ f_{k-1} \circ \ldots \circ f_1(\boldsymbol{x}), \tag{3.3}$$

with $\boldsymbol{x}^1 \equiv \boldsymbol{x}$, and the $f_i = \Phi_i(\boldsymbol{W}^i \boldsymbol{x}^i + \boldsymbol{b}_i),\ i \in \{1, \ldots, k\}$ are the individual layer functions as in Equation (3.2).

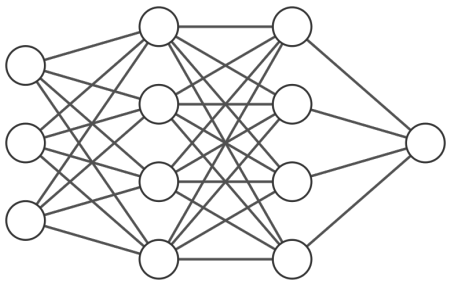


Figure 3.3: A multi-layer perceptron with a three dimensional input and a single-class output. The two central layers are 'hidden' since they are not directly observable from the outside, and are therefore called hidden layers. Figure created using LeNail[60].

3.2 Training neural networks

In contrast to other machine learning algorithms, neural networks are not trained by simply processing all the available training data just once. Instead, the training data $\mathcal{D}$ is usually divided into small batches, sometimes referred to as mini-batches, of equal size, each of which contains several dozen to a few hundred samples. This technique drastically reduces the amount of data that needs to be kept in memory simultaneously. Each batch is passed through the network in a *forward pass*, meaning that for each sample $\boldsymbol{x}$ in the batch, $F(\boldsymbol{x})$ is calculated as in Equation (3.3). The computed outputs of the batch are then compared with the *true values* (or labels) using a *loss function* as discussed in Section 2.6.1. The objective during neural network training is to minimise this loss function. Other metrics may be kept track of as well during the training, such as accuracy in case of a classification problem. If a validation set $\mathcal{D}_{\text{valid}}$ is passed to the neural network, loss, accuracy, etc. are also calculated for this set, and are subsequently referred to as validation loss, validation accuracy, etc. These values are not

used to improve network quality however, they just serve as an estimator of training success for the user.[57]

The training loss is used to update the network weights, aiming to reduce the prediction error. This can be achieved through various different optimiser functions, some of which rely on the gradients of the loss function with respect to the network weights. The updating process occurs in reverse, starting from the last layer and progressing to the first layer. Consequently, this step is referred to as *backwards propagation*, or *backpropagation*. Once all batches have been processed by the network, one *epoch of training* is complete. Fully training a neural network can take several hundred to thousands of epochs.[57]

A fixed or dynamic learning rate may be specified for the optimiser function. This parameter adjusts the magnitudes by which the weights can get updated during backpropagation. The choice of learning rate influences the convergence speed during neural network training, as well as the stability of the training process.[57]

3.3 Common structural elements of neural networks

Neural networks can consist of a few to several hundreds of layers. While most of these serve the purpose of transforming inputs in some way, other types of layers also exist. These can serve a wide variety of purposes, such as to introduce additional non-linearity, reduce or reshape the size of intermediate outputs, or regularise the learning process. The most common types of neural network elements are introduced in this section.

3.3.1 Dense layers

Multi-layer perceptrons and their layers are alternatively referred to as *fully connected* or *dense* neural networks or layers, respectively. This nomenclature stems from their weight matrices being fully connected, also known as dense. These layers treat their inputs and outputs as vectors, and require higher dimensional data to be re-shaped into 1-D vectors first. However, when working with, for instance, images, this can quickly result in huge matrices with millions of connections, potentially causing out-of-memory issues. Another drawback of vectorising data is the loss of spatial information. Fortunately, other structural elements of neural networks exist that are better suited to handling specialised input, or contribute in various ways to enhance network quality.

3.3.2 Convolutional layers

Convolutional neural networks (CNNs) are inspired by the visual cortex.[62] The term convolution stems from functional analysis, where it describes how one function modifies the shape of another function. Convolutional layers efficiently handle higher dimensional data by avoiding direct connections between inputs and outputs. Instead, a convolutional kernel (or filter) slides over the columns and rows of the input data with a pre-determined step size, known as stride.

The kernel's values are the trainable weights of the convolutional layers and are convolved (i.e. multiplied and then summed up) with the segments of the data currently 'seen' by the kernel (hence the alternative term filter). This operation calculates the point in the output layer that corresponds to the kernel's current position. Additionally, a trainable bias term that is independent of the inputs may also be added. Figure 3.4 illustrates an example of a 2D input, being processed by a 3×3 kernel with a stride of 1, without any bias. Consequently, the output is also a 2D slice. A stride of 2 would mean sliding the kernel along two cells in each step, resulting in a smaller output.[57]

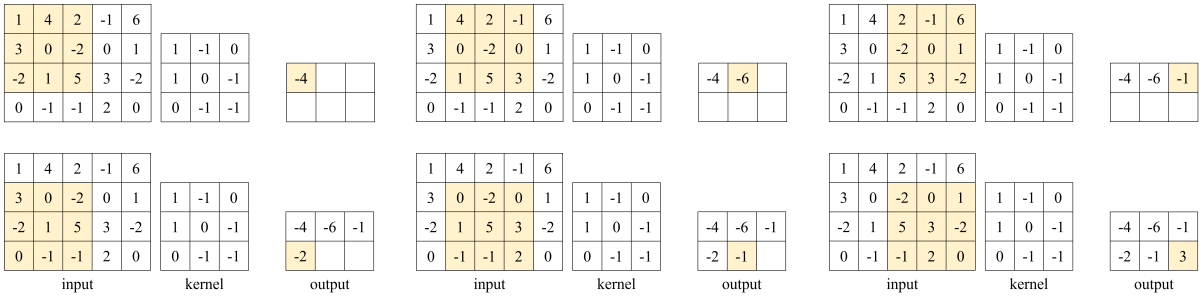


Figure 3.4: Example of a convolutional layer with a 3×3 kernel. The kernel slides along the columns and rows of the input data. Values that are seen by it (marked in yellow) are multiplied with the corresponding kernel value and then summed up. This forms the corresponding value in the output data.

The CNN example in Figure 3.4 results in a reduction of the dimensions of the input data, and causes a minor amount of information loss at the outer edges. This effect is often undesirable, necessitating so-called *padding*. Padding involves adding rows and columns of zeros around the input data, allowing the kernel to slide over the edges, as illustrated in Figure 3.5. The additional zeros do not need to be stored and will not be visible in any of the outputs.

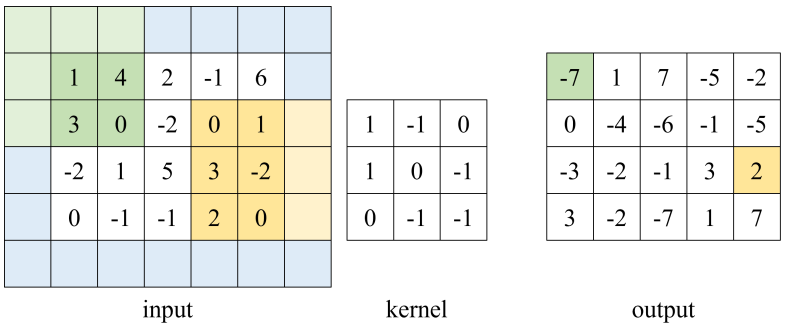


Figure 3.5: When zero padding is used, the input data are treated as if zeros were added all around, such that the filter can slide out over the edge and the input and output shape of the layer stay consistent.

In the context of computer vision, a 2D input corresponds to a monochrome image, typically

grey-scale. (Multi-)coloured images are stored as 3D data, with a separate 2D 'channel' for red, blue, and green values. Convolutional layers that handle 3D inputs therefore require 3D kernels, with one kernel slice per channel. The output in this case is a single 2D slice, where each pixel value corresponds to the sum of the individual channel/kernel slice convolutions.

Let $\boldsymbol{x}$ be the input of a convolutional layer with c channels, and $\mathbf{K}$ be a 3D kernel with a corresponding number c of slices. Then the governing equation f of a convolutional layer using $\boldsymbol{x}$ and $\mathbf{K}$ is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{c} (\mathbf{K}_i \circledast \boldsymbol{x}_i), \tag{3.4}$$

where $\mathbf{K}_i$ denotes the i^{th} slice of $\mathbf{K}$, $\boldsymbol{x}_i$ the i^{th} channel of $\boldsymbol{x}$, and $\circledast$ is the symbol for the mathematical convolution operation. Note that the bias term and activation function in were omitted Equation (3.4).

Commonly, the output of a convolutional layer will not consist of only a single channel. Indeed, modern CNN architectures often include layers with several hundred channels instead.[24,25,63,64] These typically have no more physical relation to any individual colour value etc., but usually each specialise in different aspects of image recognition such as edge detection.[24] In this case, one 3D kernel per desired number of output channels is required. Alternatively, such a kernel may be denoted as a 4D tensor $\mathbf{K}$, where the fourth dimension corresponds to the number of output channels. Input $\boldsymbol{x}$ gets convolved with $\mathbf{K}[:,:,:,j]$ to form the j^{th} output channel.

It is also possible, albeit less common, to have individual kernels per channel, that are not summed up. This type of convolution, known as depth-wise convolution[65], preserves the number of channels. Depth-wise convolutions result in fewer network parameters, which can improve training times, memory usage and inference speed, making them particularly useful for mobile applications.[66]

3.3.3 Activation functions

Equation (3.2) contained an activation function Φ, which in the case of fully connected neural networks, is responsible for introducing non-linearity into the system. Without activation functions, these networks would be equivalent to linear regression models. Common choices of activation function include piecewise linear functions such as the binary activation given by Equation (3.1), or the very popular *rectified linear unit* (ReLU)[67], defined as

$$\mathrm{ReLU}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}, & \boldsymbol{w} > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{3.5}$$

Variations of ReLU exist[68] that do not simply output zero for negative input signals, but instead perform linear (Leaky ReLU) or non-linear (exponential linear unit, ELU[69]) transformations for $\boldsymbol{x} < 0$. Highly non-linear functions such as the hyperbolic tangent tanh and various sigmoid or logistic functions are also used. These functions are illustrated in Figure 3.6. All of them

have individual strengths and weaknesses, with some being faster to compute while others add regularisation to neural network training.

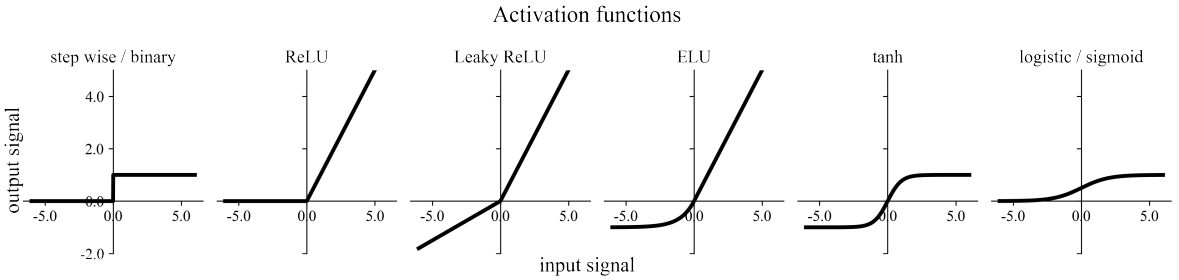


Figure 3.6: Examples of common activation functions. While some are piecewise linear (such as the step function and ReLU variations), others can be highly non-linear.

In popular deep learning libraries such as TensorFlow[70] or PyTorch[71], activation functions can be directly attached to layers like dense or convolutional layers, or included as separate layers. Similarly, modern DL architectures sometimes explicitly list activation functions or activation layers[72–74], especially if not all feature transformation layers are followed by an activation function.

3.3.4 Flatten layers

When higher dimensional data is used to predict single or multi-class outputs, at some point within the neural network graph the shape of information needs to change from being multi-dimensional to the form of a vector. This is usually achieved via so-called flatten layers, that simply vectorise the output the previous layer by stacking the data row, column, and channel wise. In particular, if input $\boldsymbol{x}$ to the flatten layer is of shape (w, h, c), its resulting output $f(\boldsymbol{x})$ will be of length $w \cdot h \cdot c$. The subsequent dense layer with n neurons therefore has a weight matrix $\boldsymbol{W}$ with $(w \cdot h \cdot c) \times n$ connections. Flatten layers can be regarded as a specialised form of reshape layer, that transform the shape of data without any further alteration.

3.3.5 Pooling layers

In general, convolutional layers can cause a significant increase in the size of data being passed through them by adding more and more channels. A common technique to counter this data growth is *pooling*[75], which originates from the *cresceptron*[76]. Pooling layers shrink the width and height of each channel by aggregating values, most often either by taking the maximum or the average of all inputs in the pooling filter. This is feasible, as individual pixels in images usually do not contain excessive amounts of information. Similar to convolutional layers, pooling works by passing a pooling filter (or pooling window) of a given width and height across the rows and columns of each data channel, moving with a specified stride.

Figure 3.7 illustrates a pooling example, where a 2×2 max pooling filter is slid over the input with a stride of 2. In this example, zero padding was added, which works exactly the same

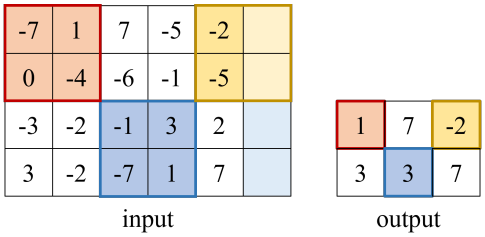


Figure 3.7: Max pooling with a 2×2 pooling filter, stride of 2, and zero padding. Without it, information from the right-most column would be lost.

as with convolutional layers illustrated in Figure 3.5. Without padding, a small amount of information, in this case the right-most column, would be cut off at the edge of the input. Note that pooling layers do not have any trainable weights at all, but are customised only by fixed hyperparameters.

3.3.6 Batch normalisation layers

Training a neural network is time and resource intensive, and can pose a number of challenges. One known issue is the fact that the loss from different batches of training data can vary significantly, requiring large changes in the network weights of each layer during the backward passes. In their 2015 paper, Ioffe and Szegedy [77] introduced a technique which they named *batch normalisation* (BN), which standardises the inputs to a network by normalising each batch. In particular, each batch is re-scaled to have a mean output of zero and standard deviation of one. Following the previous notation, the governing equation of a batch normalisation layer is given by

$$f(\boldsymbol{x}) = \gamma \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{3.6}$$

with input $\boldsymbol{x}$, μ and δ representing the mean and standard deviation of the current mini-batch, scaling factor γ and offset factor β. γ and β are the trainable parameters of these layers, and are updated during backpropagation in the same way as any other trainable weights. Moving averages of μ and σ are retained and continuously updated during training, to be used when making predictions.

Batch normalisation layers are usually inserted directly before the outputs of a layer are passed through an activation function. [77] Standardisation occurs in each batch normalisation layer, not just once per neural network. Adding such layers has been shown to accelerate training, and to provide regularisation for the network, which in turn reduces overfitting and allows for better generalisability. [78]

3.4 Advanced types of neural networks

3.4.1 Recurrent neural networks

Feed forward neural networks, as described by Equation (3.3), operate on the principle that information is sequentially passed forward through them. In particular, this means that each layer only receives the information coming from its immediate predecessor. Such networks are referred to as memory-less or acyclic, meaning that from a graph theory perspective, no cycles exist within the network graph. This suffices when working with static data such as for image recognition tasks.

However, when the data and task are time dependent or involve sequences, such as making forecasts based on current and previous events or working with language processing, it is crucial to have access to the history of a data sample. For these cases, we need *recurrent* neural networks (RNNs). These networks can include feed back connections, where information is passed back to a previous layer, as well as skip connections, where the outputs of one layer are fed into not only the subsequent layer but also to one or more additional layers further down the line. Both of these elements are illustrated in Figure 3.8. Natural language processing models for example, such as the various GPT-based ones[3] mentioned in Chapter 1, rely on RNNs to process sequences of words.

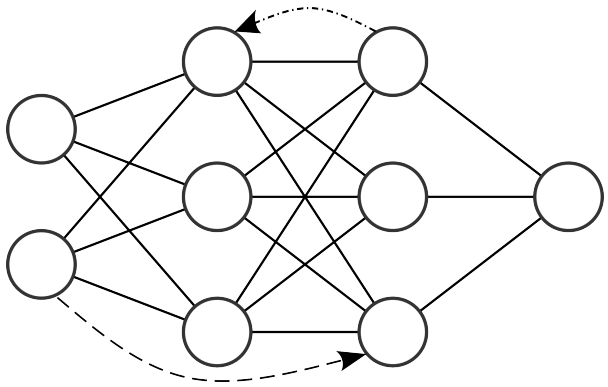


Figure 3.8: Example of a very simple recurrent neural network with one backwards connection (top, dot-dashed line) and one skip connection (bottom, dashed line). Figure created using LeNail[60].

3.4.2 Autoencoders

So far, all components of neural networks that have been introduced belong to the category of supervised learning. This implies that some form of ground truth, such as labels attached to the training data, is necessary for calculating the prediction errors during training. Autoencoders (AEs)[47] are a type of artificial neural network that instead falls into what is called unsupervised learning. The key property of AEs is that their input and output are of the same size, and the goal during training is to match these two as closely as possible.

However, in between these layers, information is passed through some form of bottleneck. The neuron count decreases (along the encoder part) and then symmetrically increases again (along the decoder part) around a central, most compressed layer, which is called the code, as shown in Figure 3.9. Autoencoders can also contain convolutional as well as other types of layers, instead of or in addition to dense layers. Their architecture still needs to adhere to the symmetrical bottleneck style presented in Figure 3.9.

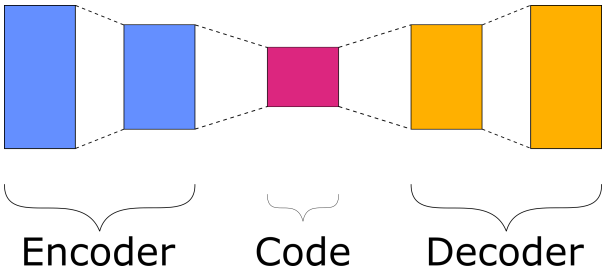


Figure 3.9: Schematic illustration of an autoencoder. Each bar represents a dense or convolutional layer, bars of same size indicate same number of neurons or channels in the respective layer. Adapted from Figure 6 in Schiessler et al.[79].

In particular, AEs can be used for dimensionality reduction and data compression, similar to a non-linear version of principal component analysis (PCA).[44–46,48,80] They can be viewed as a form of feature engineering tool that creates new features not directly from the data but from a trained model. Besides information compression, autoencoders can also be used for noise reduction[81] and anomaly detection[82].

Chapter 4

Neural architecture search

Finding optimised network topologies

In case of deep learning, hyperparameter tuning can be viewed as a two part task. First, the 'inner' hyperparameters need to be selected. These involve the number and order of topological elements that constitute the final network architecture, as well as their specific configurations. Second, the 'outer' hyperparameters must be determined. These are related to neural network training, such as the optimiser and loss functions, the learning rate, etc. As is the case in classical machine learning, the relevant selection steps can be automated. In the context of DL, researchers prefer the term *neural architecture search* (NAS)[29,30] over AutoML[52] when discussing topological concerns, i.e. the inner hyperparameters. However, the distinction between searching for either of the two parts is not always very clear, as the final output is, of course, dependent on the sum of all choices.

This chapter gives an overview over the historical development of NAS algorithms and briefly explains common elements that most of them share. Techniques for shrinking neural networks, known as *pruning*, are presented and the concept of genetic algorithms is explained.

4.1 Origins and basics

The earliest versions of what today is referred to as neural architecture search date back to at least Tenorio and Lee[83] and Moody[84]. More recently, this field was revitalised by groundbreaking works from Zoph and Le[85] and Liu et al.[86], which have led to the development of numerous extensions in various directions[30,31]. A significant challenge in Zoph and Le[85] is that their method relies on the availability of substantial computational resources, requiring the use of up to 800 GPUs over multiple weeks to achieve competitive results on the CIFAR-10[87] benchmark dataset. Weight sharing between network candidates is a commonly applied technique to reduce the computational cost of NAS algorithms.[88–92]

Most NAS algorithms share several common elements and execution steps[29,31], which are illustrated in Figure 4.1. the search space encompasses all theoretically available architectures,

typically defined by a set of rules by which these architectures can be constructed. A controller generates a large number of configurations from the search space, potentially following additional rules that may have been created or refined during previous search iterations. These configurations, also referred to as candidate architectures, are then trained on the training set $\mathcal{D}$ and ranked on the validation set $\mathcal{D}_{\text{valid}}$. An evaluation strategy is employed to provide feedback to the controller, which may update its search strategy or search space accordingly. This process is iteratively repeated until a desired stopping criterion is met. From the most current batch of candidates, a final optimised network architecture is selected. The winning network may sometimes be retrained from scratch once more before its performance is evaluated on the test set $\mathcal{D}_{\text{test}}$.

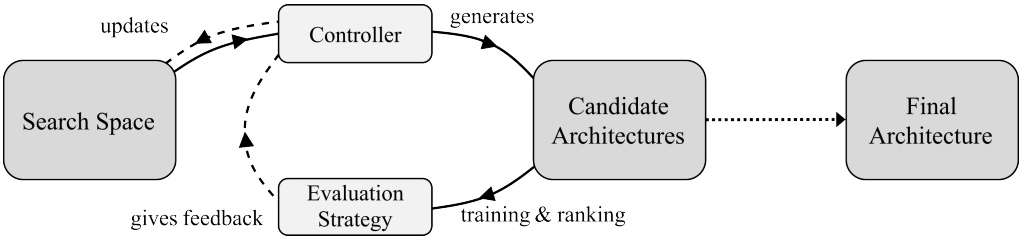


Figure 4.1: Schematic overview of the NAS process. It is repeated iteratively until a desired stopping criterion has been reached.

NAS research experienced a further popularity surge after the DARTS (*differentiable architecture search*) algorithm was published by Liu et al.[86]. Instead of working with discrete architectural components, it represents the search space in the form of a super network that contains all possible candidates, linked by what the authors call continuous operation weights. The topology selection happens via continuous relaxation and gradient-based optimisation. DARTS has a few shortcomings, such as having a high memory usage[93], and a tendency to converge to similar architectures[94]. Many publications[93–98] have since come out that have aimed to correct some of these shortcomings, and/or have extended the underlying idea in a wide range of different directions.

4.2 Genetic neural architecture search

Genetic algorithms, also called evolutionary algorithms, are inspired by the natural process of evolution. These algorithms optimise and evolve network topologies in an iterative process:[99] A population of individuals competes towards some predefined goal, such as producing the best performing neural network architecture. The fittest members are able to reproduce, and pass on (some of) their defining structural elements (genes). Offspring are created either from a single parent via mutation, or from multiple parents via recombination (i.e. cross-over between the parents' genes). This process is illustrated in Figure 4.2. The genome of a neural architecture candidate can consist of individual layers[92], or of blocks of cells, which in turn comprise of specific sequences of elements and connections[100].

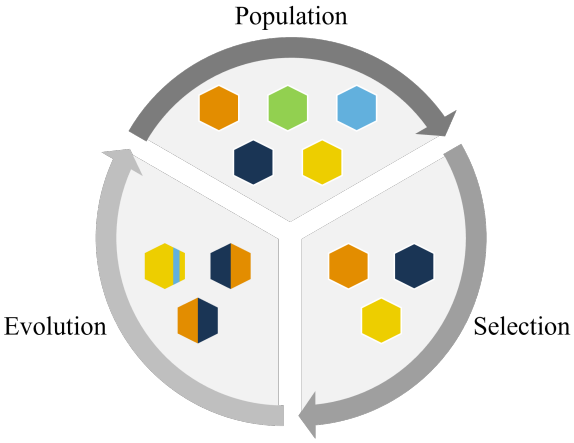


Figure 4.2: Example of a genetic algorithm. The fittest individuals of a population are selected through some fitness evaluation. They are then able to reproduce via evolutionary processes such as cross-over and mutation. The offspring form the new population generation.

In the context of neural architecture search, genetic algorithms date back to at least Miller et al.[101], who published what they called a neuro-evolutionary algorithm. Having to re-train all child networks from scratch can be quite computationally expensive.[31] Weight sharing between network candidates, i.e. re-using trainable parameters across different architectures, is sometimes employed to mitigate some of the computational costs.[88,91,92]

4.3 Pruning

The number of network parameters (i.e. trainable weights) within a neural network is a limiting factor on the complexity of tasks it can perform. If the training set is too large in relation to network's size, at some point during the training process, underfitting occurs where finer details can no longer be memorised and are constantly overwritten during backpropagation.[57] With increasing hardware capabilities, such as larger memory and faster processors, larger and more complex neural network architectures become more viable. Since improving model performance is typically their main goal, NAS algorithms tend to output rather large networks that contain millions of trainable parameters.[30]

However, there is a limit to the improvement that can be gained from increasing network sizes. As discussed in Section 2.6.2, machine learning algorithms are susceptible to overfitting, meaning they may learn idiosyncrasies and statistical artefacts found within their training data too well. This happens at the cost of a reduced ability to generalise. Large neural networks are especially prone to this behaviour, particularly when paired with comparatively small datasets.

Another downside of having a high number of network parameters is that such networks are not only costly and difficult to train, but also require extensive resources in deployment. This can be a particular challenge for applications designed to run on mobile devices such as tablets

or smart phones.

A popular approach for reducing network sizes is called *pruning*.[102–104] This term encompasses an increasing number of methods and techniques that are employed to reduce the number of network weights, either during training or for fully trained ANNs. The resulting loss in predictive quality can usually be compensated for by only a small amount of re-training.[30] Many studies have shown that network compression of up to 90% is possible without significant performance losses.[30,105]

Pruning algorithms can be roughly separated into two categories: *structured* vs. *unstructured* methods. Unstructured pruning usually involves identifying the least important network weights within the network graph and permanently setting these weights to zero. During the subsequent re-training, pruned weights are not updated any more. Structured pruning, by contrast, aims to remove entire network elements such as neurons from a dense layer or channels from a convolutional layer. To achieve this, all incoming and outgoing weights of the structure need to be removed. Figure 4.3 illustrates this distinction.

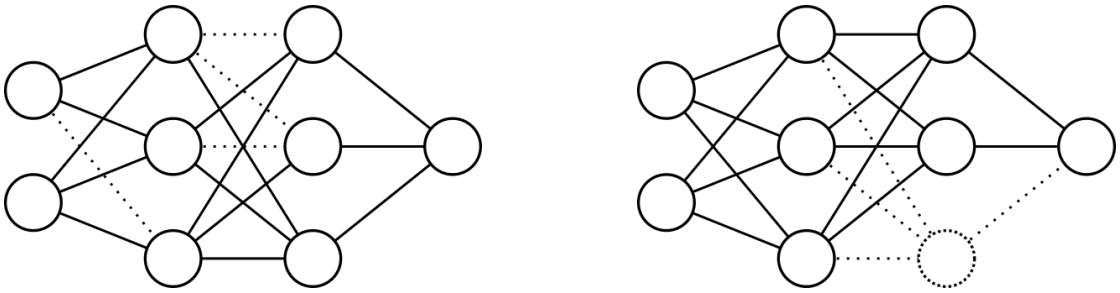


Figure 4.3: Unstructured (left) vs. structured (right) pruning. Dotted lines represent weights that have been removed or set to zero. In unstructured pruning, these can be located all over the network graph. Structured pruning however aims to remove entire units by pruning all associated incoming and outgoing weights. Figure created using LeNail[60].

Unstructured pruning is usually easier and cheaper to implement as all weights can be considered individually. Structured pruning is required however if the goal is an overall reduction of the network graph without having to rely on sparse storage techniques.[106]

Chapter 5

Examples of AI-driven research

Applying theory to practice

The previous chapters have provided a comprehensive overview over the various steps and aspects involved in the machine learning process. Each step's practical relevance can vary from case to case, along with associated challenges and difficulties. This chapter presents two use-cases in which the presented theoretical foundations are put into practice. The first one stems from the domain of material sciences, where machine learning can be used to predict how small organic compounds can alter specific properties of the light metal magnesium. The second one belongs into the field of deep learning research, where algorithms are developed that in turn facilitate the implementation and application of deep learning.

5.1 Use-case 1: Feature selection and predictive modelling for magnesium corrosion control

Magnesium is a lightweight metallic element that occurs abundantly on Earth[4] and exhibits a relatively high electrochemical reactivity[5]. It holds promise for a wide range of highly versatile applications. These can be found in transport sectors like the automotive and aerospace industries[6–8], for which corrosion needs to be prevented. In a medical context, magnesium finds application in, for example, biodegradable implants[11,12], where degradation rates need to be adjusted to the specific circumstances of the injury. Magnesium furthermore serves as an anode material in batteries[9,10], where a constant degradation rate is required. Hence, precise control of its corrosion behaviour is necessary to meet varying demands.

Such control can be gained via the addition of small organic molecules to either a coating matrix, or in the case of batteries, directly to the electrolyte.[13–15] These molecules can be described by thousands of distinct attributes, stemming either directly from the compound's electrochemical and structural properties, or derived from multi-scale computer simulations[107,108] (such as molecular dynamics or density functional theory). Which attributes are most pertinent to the molecule's influence on magnesium corrosion is not immediately apparent. The search for

suitable contenders is further complicated by the fact that the chemical space from which these molecules may be selected is almost unlimited.[16] It cannot be effectively explored with even the most capable high-throughput experimental testing setups.

Consequently, data-driven modelling is required to identify suitable organic compound candidates. Machine learning algorithms such as the various feature selection methods presented in Section 2.2 are especially powerful tools for extracting the relevant information from such intricate and complex environments. With their aid, a number of molecular attributes or features can be identified that are highly relevant with respect to a compound's influence on magnesium corrosion behaviour. Subsequently, predictive models relying on this feature set can be trained to help search for materials with promising capabilities.

5.1.1 Identifying relevant molecular descriptors

A database containing measurements of the corrosion behaviour of a specific magnesium alloy called ZE41[14], when paired with different small organic compounds, served as the starting point for **publication 2**. From this database, 60 different chemical compounds of interest were selected.

The target variable in the resulting dataset $\mathcal{D}$ is *inhibition efficiency* (IE). This percentage value describes how effective the organic compound is at inhibiting corrosion compared to that of the pure alloy ZE41. Positive values indicate a slowdown in corrosion (with an IE of +100% corresponding to no corrosion at all), whereas negative values signify an acceleration of corrosion. The target variable values are distributed unevenly across a range of $[-270, 75]$ IE / %, as illustrated in Figure 5.1.

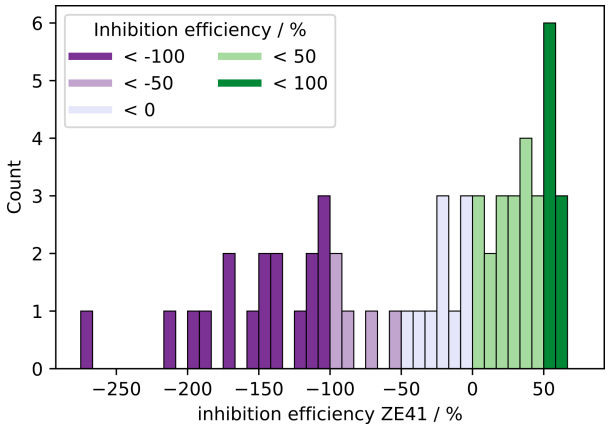


Figure 5.1: Distribution of the inhibition efficiency values of the 60 small organic compounds used in **publication 2**. The data are spread unevenly across the range of $[-270, 75]$ IE / %. Previously unpublished figure.

The compounds can be described by thousands of attributes derived either directly from structural and electrochemical properties, or calculated using various quantum chemical and chemin-

formatics software packages. In the context of magnesium corrosion research, the term (molecular) descriptor is used synonymously with feature. Only descriptors that were not constant or almost zero across the entire dataset were retained, resulting in a total of 1,260 features per sample.[79] From a machine learning perspective, this dataset is highly unsuitable in its current state, with a small sample size compared to an excessive number of attributes.

After splitting off 10% (i.e. 6) of the 60 samples as a representative validation set $\mathcal{D}_{\text{valid}}$ and normalising the feature ranges using a min-max-scaler, three different feature selection strategies were employed (c.f. Section 2.2). These were used to select the top 3, 5, and 63 (i.e. 5% of all available) features.

1. Analysis of variance (ANOVA)
2. Recursive feature elimination (RFE) with random forests (RF) as underlying classifier
3. Random sampling

F-score based ANOVA is a deterministic scoring method and was thus calculated only once, see also Section 2.2. More than 90% of all available features reached a score of less than 20% of the maximal score, as shown in Figure 5.2 (left). The sets of n best-performing descriptors based on ANOVA were determined by taking the respective top n entries from the sorted list.

Random forest-based recursive feature elimination yields results that vary based on the chosen random seed, see also Section 2.2. Therefore, all RFE-based selection variants (i.e. searching for the top 3-, 5- and 63-tuples of descriptors) were repeated 100 times each with different random seeds. From these repeated runs, the most frequently occurring top 3- and top 5-tuples were identified. No best 63-tuple composition was selected more than once, however. A resulting 'best performing' 63-tuple was artificially based on the frequency with which each descriptor was included in any of the identified 63-tuples, as shown in Figure 5.2 (right). Out of 504 unique molecular descriptors, the vast majority ($\approx 60\%$) were selected at most 5 times, and only 5 features were selected during each run.

Both feature selection methods indicated that the vast majority of the 1,260 features are of little importance, as illustrated in Figure 5.2. Notably, the lowest unoccupied molecular orbital (LUMO), a density functional theory (DFT)-derived descriptor, consistently emerged as highly relevant in each method, which seems to confirm previous findings[18] based on human intuition.

5.1.2 Predictive modelling

To evaluate the predictive abilities of the selected features, each of the identified subsets were used as sole inputs for neural networks. We compared the root mean squared errors (RMSE, see Equation (2.7)) of predictions on the representative validation set $\mathcal{D}_{\text{valid}}$ withheld before beginning the feature selection and training process. As additional baselines, we trained neural networks using randomly selected groups of features, as well as on the whole set of 1,260 molecular descriptors. The neural network training process was repeated 100 times per selection method and number of features to counter statistical artefacts from underlying random seeds, and predictive results were averaged over all models of the same type. Among all these

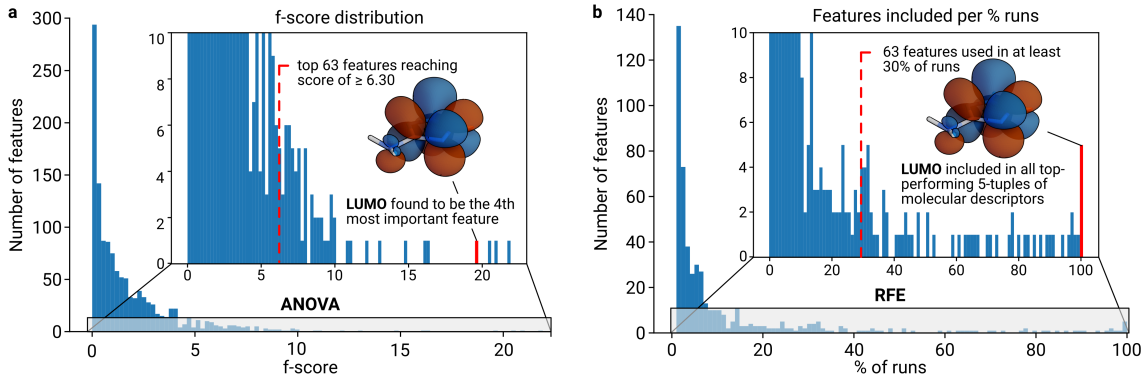


Figure 5.2: Feature importance distributions for magnesium dissolution modulators as identified in **publication 2**. a: Feature count per f-score value, determined by the ANOVA filter method. b: Frequency of inclusion for 504 unique descriptors in 100 repeats of RFE-based selection, identifying the best-performing 63-tuple of features. Adapted from Figure 2 in Schiessler et al. [79].

variations, the 5-tuple sets of descriptors selected using the RFE method performed best. This result seems reasonable given that RFE is able to capture the correlations and dependencies between groups of features instead of regarding each one on its own.

Since the available dataset $\mathcal{D}$ of 60 organic compounds is very small in terms of machine learning, we could demonstrate that only a few outliers had a significant influence on the overall performance of our methods. After excluding a compound that, uniquely within $\mathcal{D}$, contains a certain chemical structure, the neural networks trained on the RFE-based 5-tuple of descriptors achieved an average RMSE of only 26pp (percentage points) on a representative validation set $\mathcal{D}_{\text{valid}}$, as illustrated in Figure 5.3. An error of 26pp is well within the observed error margin of experimental results found in e.g. Lamaka et al. [14]. These results indicate that even given the limited amount of available data, feature selection methods are able to identify molecular descriptors of small organic molecules that can be used to understand and predict the compound's influence on the corrosion behaviour of the magnesium alloy ZE41.

5.1.3 Autoencoders for outlier detection

As a further point of interest, we investigated how autoencoders might be used for outlier detection. This special type of neural network is explained in more detail in Section 3.4.2. It can be used to generate a lower-dimensional representation of the available features, somewhat similar to a non-linear version of PCA. When coupled with a predictive model, the decoder part of the autoencoder can be used, for example, to generate a contour map of the target variable's predicted values across the space spanned by the code. Plotting the true IE values of the training and validation samples onto this contour map can be used to identify compounds which are significantly mis-predicted, as illustrated in Figure 5.4.

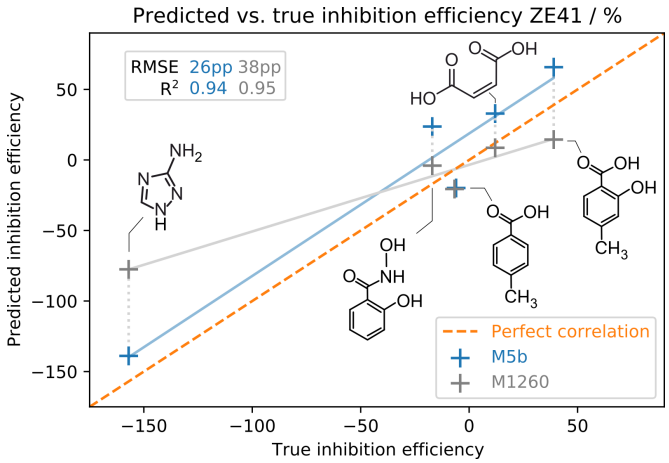


Figure 5.3: Predicted vs. true inhibition efficiency values for samples in the representative validation set $\mathcal{D}_{\text{valid}}$ in **publication 2**. M5b indicates results from neural networks trained on the 5-tuple of features identified via RFE, M1260 those from neural networks trained on the full feature set. Published as Figure 3 in Schiessler et al.[79].

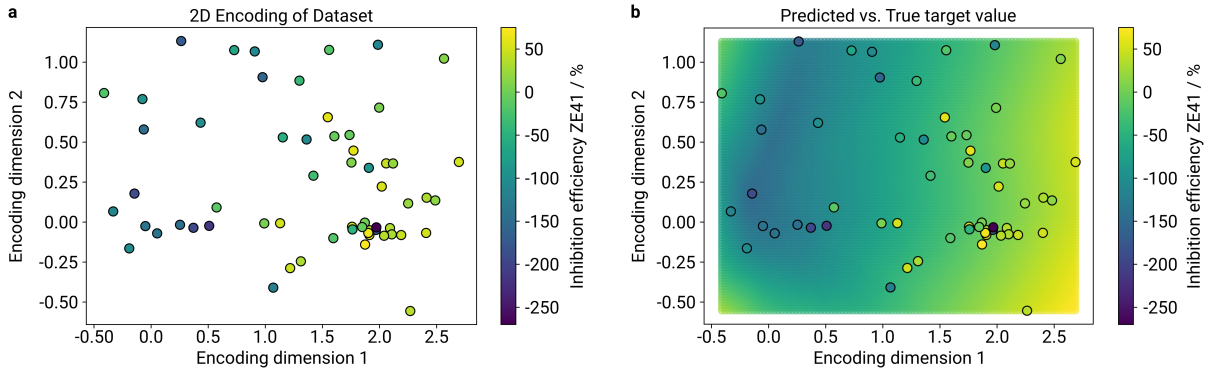


Figure 5.4: a: Samples from $\mathcal{D}$ in **publication 2** plotted in a reduced 2-dimensional code space. b: A contour map of the 2-d space spanned by the code can be created using the decoder part of the autoencoder in combination with a predictive model. Published as Figure 4 in Schiessler et al.[79].

5.1.4 Validation of results

When applying machine learning methods on such small datasets, there is a high risk of overfitting on the available training data, which results in models that perform poorly on previously unseen samples. To assess the robustness of the methods and predictive capabilities established in **publication 2**, a blind testing dataset $\mathcal{D}_{\text{test}}$ comprising 15 previously untested small organic compounds was created using the ExChem routine[22], as illustrated in Figure 5.5. These formed the blind testing set $\mathcal{D}_{\text{test}}$. The same experimental setup was used as for the original dataset $\mathcal{D}$ of 60 magnesium dissolution modulators published by Lamaka et al.[14]. The combined dataset,

$\mathcal{D}_{\text{comb}}$, comprises both $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$.

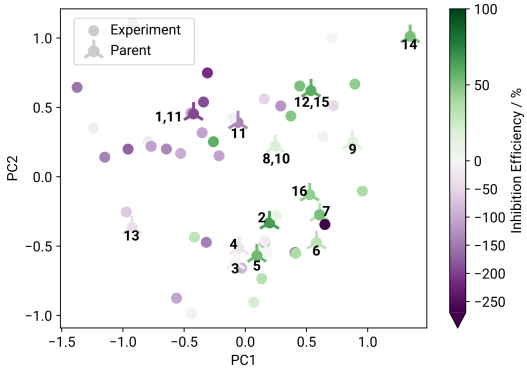


Figure 5.5: Structure-property landscape of the 60 magnesium dissolution modulator samples in $\mathcal{D}$. 20 compounds served as 'parents' (crossed circles) for generating the blind test set $\mathcal{D}_{\text{test}}$ for **publication 3**. Published as Figure 1 in Schiessler et al. [109].

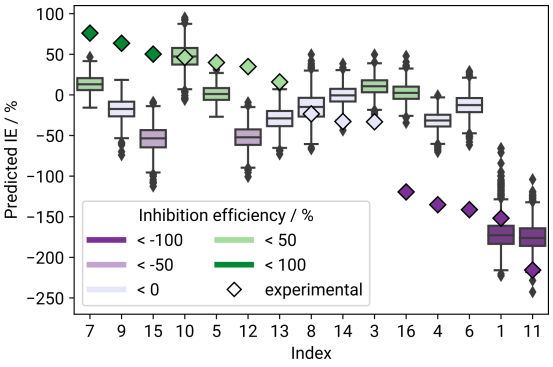


Figure 5.6: Distribution of predictions on $\mathcal{D}_{\text{test}}$ in **publication 3**. Boxes are coloured according to the compound's mean predicted IE values. Respective experimental IE values are represented as coloured diamonds. Published as Figure 2 in Schiessler et al. [109].

We replicated the feature selection process, prioritising RFE to identify 5-tuples of molecular descriptors, as this variant showed optimal performance in **publication 2**. Selected features for $\mathcal{D}_{\text{comb}}$ consisted of a 3 : 2 combination of those chosen for $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$, reflecting the proportion of samples per dataset. This showcases the robustness of our selection method under extended data availability. Notably, there was no overlap between the identified 5-tuples for $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$, indicating a somewhat limited ability of the models trained for **publication 2** to capture the specific qualities of samples in $\mathcal{D}_{\text{test}}$.

This expected reduced generalisability was confirmed by predictive results on $\mathcal{D}_{\text{test}}$ using the models from **publication 2**. Boxplots of the prediction distribution per sample are illustrated in Figure 5.6. Approximately half of the 15 samples in $\mathcal{D}_{\text{test}}$ were predicted within acceptable error margins, with 6 of the remainder being extreme outliers. A structure-property landscape projection, shown in Figure 5.7, revealed that mis-prediction of these compounds occured most likely due to two main reasons:

Compounds 4, 6 and 16 (c.f. Figure 5.6), experimentally determined to be strong corrosion accelerators, are located in a cluster of similarly structured, but experimentally widely varying samples. Additionally, their measured inhibition efficiencies are on the lower end of the overall range, where few samples exist, as can be seen in Figure 5.1.

Compounds 9, 12, and 15 (c.f. Figure 5.6), projected into a region containing both weak accelerators and weak inhibitors in Figure 5.7, are experimentally determined to be weak to moderate inhibitors. The selected features seem to be less suited for capturing the specific properties of these compounds.

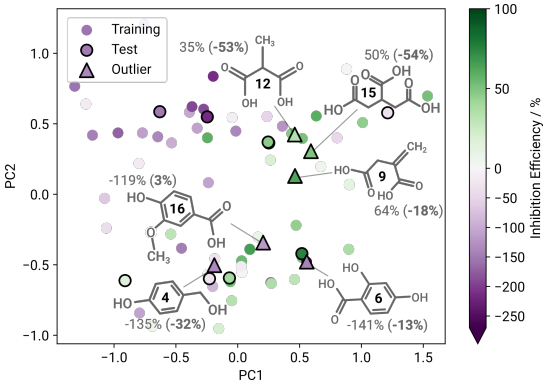


Figure 5.7: Structure-property landscape for samples in $\mathcal{D}$ from **publication 2** and $\mathcal{D}_{\text{test}}$ from **publication 3**. Compounds that were identified as extreme outliers within $\mathcal{D}_{\text{test}}$ are marked, along with their measured (**predicted**) IE values. Published as Figure 4 in Schiessler et al.[109].

While the models trained for **publication 2** might not yet generalise for any new type of compound, they show promise. The methods used in **publication 2** scale well when new data are added to the existing pool. Predictive outliers identified in **publication 3** provided further insights into structural regions that particularly benefit from additional samples.

5.2 Use-case 2: User-friendly and lightweight genetic neural architecture search

At only 60 samples, the dataset $\mathcal{D}$ that was used throughout Section 5.1 is small in terms of machine learning. In relation to standard deep learning datasets, which usually contain at least several thousand samples, it is outright tiny. Training neural networks on such a limited amount of data poses the risk of severe overfitting due to massively overparameterised network architectures. The training process itself, on the other hand, advances fairly quickly, since very few samples need to be passed through the network. This makes manual topology selection feasible, as long as the user has a general idea what options to select from.

With a rising number of samples and/or an increasing complexity of tasks, manual optimisation becomes slower, more difficult, and more costly in terms of computational budget. Neural architecture search methods, presented in Chapter 4, are a powerful tool to aid in the construction and selection of suitable network topologies. However, many NAS algorithms rely on repeatedly training numerous different candidates.[31] While weight sharing can mitigate some of the resulting computational costs[89], NAS methods based on such techniques often require pre-trained meta-controllers, and/or generate extensive network graphs that represent multiple individual networks simultaneously.[86,94] Therefore, despite being powerful in identifying suitable network topologies, these algorithms often necessitate extensive computational resources, considerable user experience, or, in the worst case, both. Moreover, such algorithms tend

to select relatively large, overparameterised network architectures[30]. This may be especially unsuitable for certain applied scientific fields where data gathering is a limiting factor. Lightweight and user-friendly solutions that can handle small datasets are therefore a huge asset for researchers working in such fields.

5.2.1 Functionally equivalent, structurally different networks

The main idea behind the NAS algorithm presented in **publication 1** was to identify functions that are capable of altering the structure of a neural network while maintaining its input-output behaviour. In mathematical terms, if $F(\cdot)$ is the governing equation for a given network, we searched for $F'(\cdot)$ such that

$$F(\boldsymbol{x}) \equiv F'(\boldsymbol{x}) \; \forall \; \boldsymbol{x}, F \neq F'. \tag{5.1}$$

With such functions, we aimed to create a genetic neural architecture search algorithm, where functionally equivalent but structurally different offspring compete against each other. These topological variations are introduced in the hope of overcoming local optima during the training process. The associated fitness function takes into account the candidate's predictive capability as well as an estimate of it's potential to improve.

For this first prototype, we focused solely on fully connected neural networks, where weight matrices directly represent the existing connections within the network graph. For such matrices, the addition of structural elements without a change in the overall network behaviour is easily possible under certain conditions.[110] New layers are initialised using an appropriately shaped identity matrix as weight matrix. Let $F(\boldsymbol{x}) = f(\boldsymbol{x}) = \Phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ represent a single dense layer as in Equation (3.2). Then

$$F'(\boldsymbol{x}) = f' \circ f(\boldsymbol{x}) = \boldsymbol{I}(\Phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})) + \boldsymbol{0} = \Phi(\boldsymbol{I}\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \Phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = F(\boldsymbol{x}), \tag{5.2}$$

with weights given by the identity matrix $\boldsymbol{I}$ and zero-vector $\boldsymbol{0}$ as bias term. Equation (5.2) holds true, iff the activation Φ is at least piecewise linear, which is met by e.g. the ReLU activation given by Equation (3.5). Neurons are added to existing layers by copying some of the existing weights, and multiplying them with a scaling factor. These adaptations form the backbone of the Net2Net transformations introduced by Chen et al.[110] that aim to rapidly train large neural networks with the aid of pre-trained smaller networks.

However, to avoid exclusively creating networks of increasing size, we also needed methods to compress network architecture that preserve as much functionality as possible. Singular value decomposition (SVD), a tool from linear algebra, can be used to find projections onto smaller-dimensional sub spaces. It has been successfully employed in structured pruning.[111–113] Following the Eckart-Young(-Mirsky) theorem[114], SVD allows us to find the closest possible lower rank representation under the Frobenius norm. This means that when creating a smaller (weight) matrix to replace the original one via SVD, we are able to ensure that only the least important bits of information are discarded.

5.2.2 The Surgeon

These network adaptations formed the pool of available mutations for a genetic neural architecture search algorithm that we named 'the Surgeon', a nod to Mary Shelley's *Frankenstein*[115]. Instead of relying on a pre-trained or heuristic meta-controller, we implemented a deterministic recommendation module for deciding which modifications should be applied. It combines an analytical analysis of the weights of each layer that uses SVD to identify how many, if any, neurons are superfluous, with a statistical estimation of network performance under the given modification, which was derived from the Bayesian information criterion (BIC)[116].

Next to improving predictive capabilities, an additional objective for the Surgeon is to balance network efficacy with network size. Therefore, the Surgeon's fitness function weighs candidate performance against performance increase and parameter count difference with regard to the candidate's parent network. Let a denote the candidate's validation accuracy, Δa the accuracy increase and p the parameter size fraction compared to the respective parent's values. The fitness function is therefore given by

$$\text{fitness} = a + \frac{\exp(\Delta a)}{\exp(p)}. \tag{5.3}$$

This function punishes networks that grow too large, while rewarding those that show high potential for further improvement.

The Surgeon is implemented in the style of a genetic algorithm. It alternates training and optimisation phases, and may retain several competing topologies simultaneously, which are called branches. Modifications are created for each branch during the optimisation phase, and pooled to form the set of candidates for the current generation. The fitness function given by Equation (5.3) selects the n best performing children to form the branches of the new parent generation. Algorithm 1 (adapted from Algorithm 1 in Schiessler et al.[92]) depicts this process.

5.2.3 Evaluation of results

The Surgeon's performance was evaluated on various standard imaging benchmark datasets, employing three different starting topologies. Experiments were repeated multiple times per dataset/ starting topology combination and then averaged. As a baseline, we compared the Surgeon's performance to results obtained by repeatedly training the provided starting topology for the same number of epochs.

For all dataset and starting topology combinations, the validation accuracies of the final topologies selected by the Surgeon reached or exceeded those of the baseline. In cases where the naive validation accuracy was already high, the relative accuracy increases by the Surgeon where modest, as illustrated in Figure 5.8. However, our algorithm was able to reduce the network size without significant performance loss in several cases. In instances where the baseline's performance was low or where it was not learning at all due to an insufficient network size, the Surgeon was able to overcome the initial local minimum and significantly improved network performance, as illustrated in Figure 5.9.

Algorithm 1 The Surgeon

function RUN(initial topology t_0)
  pre-train t_0 (warm start)
  initialise parent generation $P_0 = \{t_0\}$

  while termination criteria not met **do**
    identify all potential children C_{pot} from all parents $p \in P_i$
    repeat
      score C_{pot} and select candidates C_{sel}
      re-train all candidates in C_{sel} for a small number of batches
      evaluate candidates in C_{sel}
      keep n top scoring ones as child generation C_i
      train all children $c \in C_i$ until full epoch step
    until improvement achieved or max re-tries reached
    child generation C_i becomes new parent generation P_{i+1}
  select final best scoring individual t_{opt} from P_{final}
  return fully trained optimised network t_{opt}

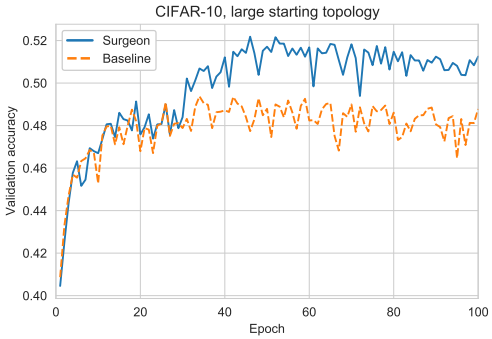


Figure 5.8: Example of a single run of the Surgeon on the CIFAR-10 dataset[87] in **publication 1**. At around 30 epochs, the Surgeon manages to overcome a local optimum. Published as Figure 4 (left) in Schiessler et al.[92].

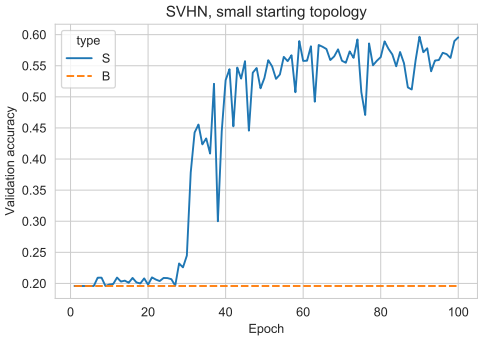


Figure 5.9: Example of a single run of the Surgeon on the SVHN dataset[117] in **publication 1**. The Surgeon is able to rescue an insufficient topology. Previously unpublished data.

On average, the Surgeon required approximately 290 epochs of training per dataset and starting topology to reach an equivalent of 100 epochs of naive baseline training. Our experiments made apparent that, in most cases, this number of epochs exceeds what is needed until convergence is reached. With appropriate early stopping techniques or a more dynamic training schedule, the required computational costs can be significantly reduced[118], making it a computationally inexpensive version of a neural architecture search algorithm.

5.2.4 Extending the capabilities of the Surgeon

A limitation of the Surgeon is its restriction to feed forward neural networks, i.e. dense layers linked strictly in sequence. While such networks are easiest to analyse, understand, and modify, their capabilities are also somewhat limited. Thus, for **manuscript 4**, we implemented an extension of the Surgeon capable of handling convolutional layers in addition to plain feed-forward neural networks. The resulting algorithm was named 'ECToNAS' (evolutionary cross-topology neural architecture search). A key aspect is its ability to independently identify which type of network topology (i.e. fully connected or convolutional) is best suited for a given task and dataset.

Introducing new convolutional layers or adding channels to existing ones can again be achieved using the Net2Net transformations published by Chen et al. [110]. The removal of channels from a convolutional layer or entire convolutional layers is more complicated however than the removal of whole or parts of dense layers. For these, we adapted a method for structured pruning [119] that utilises information from batch normalisation layers, which are often paired with convolutional layers. In order to limit network size, pooling layers are always added after convolutional layers. ECToNAS therefore uses what we name 'convolutional cells', which always consist of a convolutional, pooling, batch normalisation, and activation layer in this order.

ECToNAS does not rely on deterministic rules to decide which offspring are created per generation, instead all potential network adaptations are performed. The following survival-of-the-fittest competition happens in two phases. First, the best competitors per modification type are identified in individual bracket-style tournaments. These then enter a second tournament that yields the fittest offspring out of the whole generation. This selection process is illustrated in Figure 5.10.

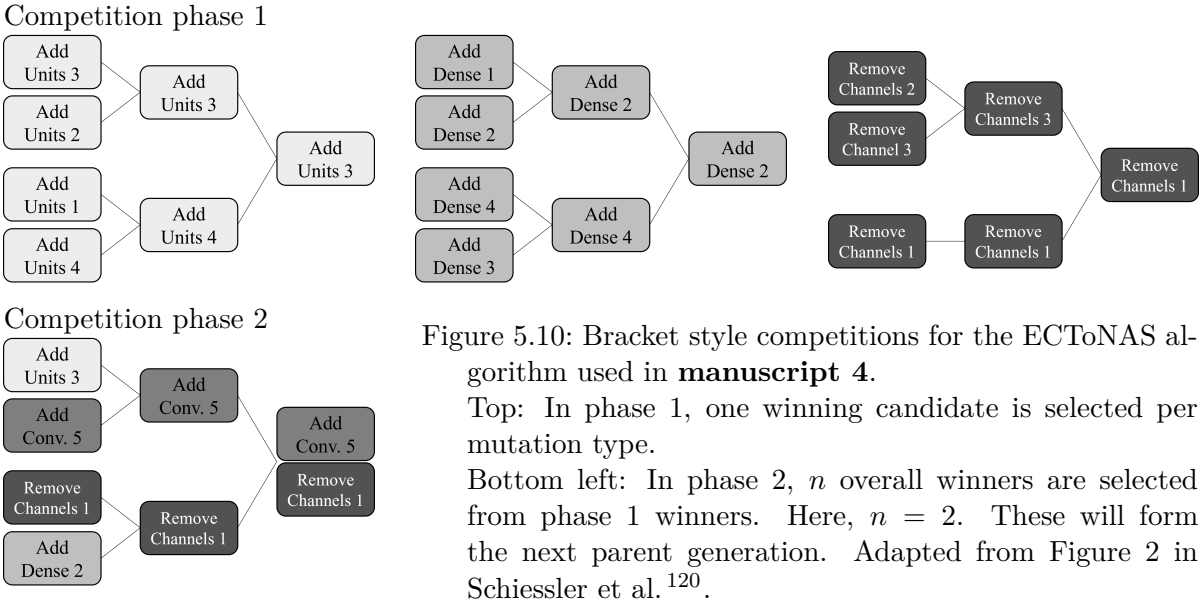


Figure 5.10: Bracket style competitions for the ECToNAS algorithm used in **manuscript 4**.
Top: In phase 1, one winning candidate is selected per mutation type.
Bottom left: In phase 2, n overall winners are selected from phase 1 winners. Here, $n = 2$. These will form the next parent generation. Adapted from Figure 2 in Schiessler et al. [120].

We handed additional control over ECToNAS's optimisation behaviour to the user via a tune-

able hyperparameter that toggles between 'greedy' and 'balanced' modes. In the former, the algorithm greedily tries to maximise network performance, while accepting arbitrarily large network sizes. In the latter, network parameter count is weighted against network performance in a manner similar to that of the Surgeon.

5.2.5 ECToNAS experiments

ECToNAS performance was evaluated on several standard imaging benchmark datasets, using three different starting architectures and two different topology types (simple feed forward vs. convolutional neural networks). Experiments were repeated multiple times and then averaged per starting configuration. Straight-forward training of the naive starting topologies for a fixed amount of epochs served as a baseline for comparison. Additionally, the experiments were repeated with a deactivated fitness function, where winners for the bracket-style competitions (c.f. Figure 5.10) where randomly determined. We denoted these runs as random mode.

In greedy mode, ECToNAS consistently outperformed the respective baselines and the results from random mode. The parameter count of the identified winning topologies increased by a small to moderate amount. As expected, balanced runs of ECToNAS on average scored worse than greedy runs by around 5-10 percentage points. Notably, however, the balanced runs nearly all performed at least as well as the baseline while achieving network compression rates between 80% and 90%. An example of a balanced run of ECToNAS is illustrated in Figure 5.11. Its performance matches that of the baseline, while parameter count drastically decreases.

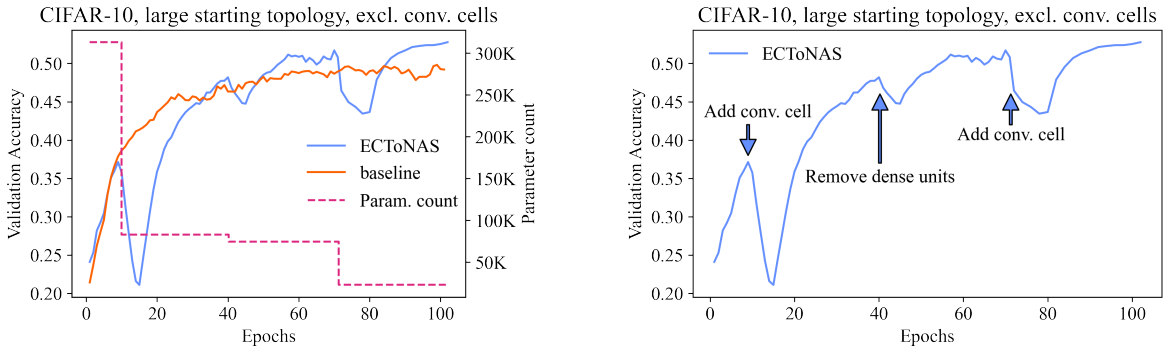


Figure 5.11: Example of a single run of the ECToNAS algorithm on the CIFAR-10 dataset[87] in balanced mode in **manuscript 4**. Left: Comparison with baseline performance and development of network parameter count. The final validation accuracy of ECToNAS surpasses that of the baseline, while the network is compressed to around 7% of its original size. Right: Annotation of performed modifications. Included as Figure 6 in Schiessler et al.[120].

Additionally, a use-case was artificially generated that was intended to mimic data of unclear or mixed types or objective functions. For this, a tabular dataset was adapted and reshaped to appear as image data. Inexperienced users might be tempted to include one or more convolutional layers that are not required, as no structural information is present. Such layers

may even be detrimental to the network's performance as overfitting may occur on presumed correlations within the data that in truth do not exist. In greedy mode, several runs of the ECToNAS algorithm were indeed able to identify this situation correctly and to remove all present convolutional cells. This behaviour is depicted in Table 5.1. We take this as proof of concept that ECToNAS is able to independently perform topology crossings both towards and away from convolutional neural networks.

Starting topology	CNN			FFNN		
Final topology	CNN	FFNN	Crossing	CNN	FFNN	Crossing
Greedy ($\alpha = 1$)	70	5	6.7%	0	75	0.0%
Non-greedy ($\alpha = 0.5$)	75	0	0.0%	67	8	89.3%
Non-greedy ($\alpha = 0$)	75	0	0.0%	74	1	98.6%

Table 5.1: Counts of resulting topology types per starting topology type and greediness level, as well as percentage of runs that ended in topology crossing. Included as Table 3 in Schiessler et al.[120].

Similar to the Surgeon, ECToNAS is a user-friendly, cost-effective genetic neural architecture search algorithm. It is able to select suitable network architectures for a given task and dataset, and can choose and switch between different types of topologies even at runtime.

Chapter 6

Summary and discussion

Connecting the dots

This thesis was written as a comprehensive guide for both theoretical and applied machine learning, offering insights into the most relevant practical aspects of the machine learning process. I provided a detailed walk-through, and presented various tools and techniques along with their respective backgrounds and related theories. The demonstrated steps include preparing datasets, extracting relevant information, choosing well-fitting algorithms along with their hyperparameters, and assessing the quality of obtained results. Algorithmically optimising all the choices and settings related to this process is called AutoML, which constitutes its own research field.

As the complexity of the tasks increases, so do the demands on applied algorithms. To address this, I introduced artificial neural networks, the most powerful class of ML algorithms available today. ANNs are more flexible than classical ML algorithms, and can consist of many different types of layers and connections. Like human brains, they possess plasticity, allowing them to incorporate new information once it becomes available with having to re-start the entire training process. However, optimising neural networks for a specific task and dataset can be a challenging and time-consuming process. In the context of ANNs, automating the search for the best configuration is called neural architecture search.

Within this thesis, I presented two use-cases that successfully apply the above methods and concepts. I detailed an example from theoretical machine learning research, where we developed two iterations of a genetic neural architecture search algorithm. They were built with the intent of being very light-weight and user-friendly, and capable of handling limited amounts of training data. The fundamental idea was to modify existing network architectures by creating structurally different but functionally equivalent ANNs. These candidates then underwent training and performance evaluation, allowing the identification of topologies that may overcome local optima during the training process. The first prototype, named the Surgeon, served as a proof-of-concept and was validated using several image classification benchmark datasets. It managed to rescue insufficient topologies, increased network performance, and, in some cases, compressed network sizes simultaneously. Its successor, ECToNAS, added support for a wider

range of structural elements and can independently switch between different topology types at runtime. Both algorithms were designed to aid even inexperienced users, enabling them to successfully employ deep learning methods in their respective domains.

Furthermore, I discussed an example from material sciences, where machine learning aided in understanding and controlling the corrosion behaviour of the metal magnesium. Using a dataset of 60 small organic compounds, we employed feature selection methods to identify the most relevant molecular attributes. Our findings confirmed previous results based on human intuition. The selected features served as inputs for training ANNs, predicting corrosion inhibition efficiency with an error margin comparable to experimental measurements. Validation using previously untested compounds confirmed the robustness and scalability of our approach.

6.1 Outlook

Scientific work is a continuum, and even the best results can always be extended or improved. This of course holds true for my own work as well. Both the autoencoder approach mentioned in Section 5.1.3, as well as the validation process detailed in Section 5.1.4, identified regions within the predictive domain that are challenging to grasp for our magnesium corrosion behaviour models. Generating additional data from samples that are situated in these specific areas has the potential to greatly benefit the generalisability of our predictions.

The neural architecture search algorithms from Section 5.2 could, in turn, be extended by adding support for more diverse layers or more complex types of connections, allowing, for example, recurrent connections to occur within the network graph. A more dynamic search strategy could increase efficiency and further decrease overall runtime and computational costs.

Ultimately, I would like to bring both use-cases together and integrate everything into one solid pipeline. In this ideal scenario, the data preprocessing and feature selection methods analyse new samples, and if required, update the set of most relevant features. The subsequent deep learning model is then either re-trained from scratch or simply receives some additional fine-tuning. Finally, the autoencoder can be used to identify the next areas that could benefit the most from additional samples. This idea is illustrated in Figure 6.1. Thus, whenever new data becomes available, integrating this new knowledge at last becomes as simple as clicking a button.

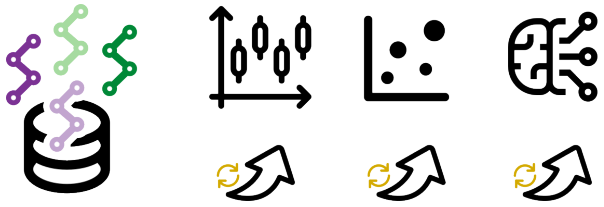


Figure 6.1: Schematic of a fully automated workflow. Once new data becomes available, they are automatically used for making predictions, identifying outliers, and running the Surgeon or ECToNAS to update the predictive model as required.

Bibliography

[1] R. Fjelland. Why general artificial intelligence will not be realized. *Humanities and Social Sciences Communications*, 7(1):10, June 2020. 10.1057/s41599-020-0494-4.

[2] Future of Life Institute. Policymaking in the Pause. https://futureoflife.org/open-letter/pause-giant-ai-experiments/, March 2023. https://futureoflife.org/open-letter/pause-giant-ai-experiments/. Accessed on: 2023-08-08.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[4] D. L. Anderson. Chemical composition of the mantle. *Journal of Geophysical Research: Solid Earth*, 88(S01):B41–B52, 1983. 10.1029/JB088iS01p00B41.

[5] T. Würger. *A computational approach to magnesium corrosion engineering*. PhD thesis, Technische Universität Hamburg, Hamburg, 2023. https://doi.org/10.15480/882.4920.

[6] A. I. Taub and A. A. Luo. Advanced lightweight materials and manufacturing processes for automotive applications. *MRS Bulletin*, 40(12):1045–1054, December 2015. 10.1557/mrs.2015.268.

[7] A. Dziubińska, A. Gontarz, M. Dziubiński, and M. Barszcz. The Forming of Magnesium Alloy Forgings for Aircraft and Automotive Applications. *Advances in Science and Technology Research Journal*, 10(31):158–168, September 2016. 10.12913/22998624/64003.

[8] W. J. Joost and P. E. Krajewski. Towards magnesium alloys for high-volume automotive applications. *Scripta Materialia*, 128:107–112, 2017. 10.1016/j.scriptamat.2016.07.035.

[9] T. Zhang, Z. Tao, and J. Chen. Magnesium-air batteries: from principle to application. *Materials Horizons*, 1(2):196–206, 2014. 10.1039/C3MH00059A.

[10] M. Deng, L. Wang, D. Höche, S. V. Lamaka, P. Jiang, D. Snihirova, N. Scharnagl, and M. L. Zheludkevich. Ca/In micro alloying as a novel strategy to simultaneously enhance power and energy density of primary Mg-air batteries from anode aspect. *Journal of Power Sources*, 472:228528, October 2020. 10.1016/j.jpowsour.2020.228528.

[11] H. S. Brar, M. O. Platt, M. Sarntinoranont, P. I. Martin, and M. V. Manuel. Magnesium as a biodegradable and bioabsorbable material for medical implants. *JOM Journal of the Minerals Metals and Materials Society*, 61(9):31–34, September 2009. 10.1007/s11837-009-0129-0.

[12] B. J. C. Luthringer, F. Feyerabend, and R. Willumeit-Römer. Magnesium-based implants: a mini-review. *Magnesium Research*, 27(4):142–154, October 2014. 10.1684/mrh.2015.0375.

[13] S. V. Lamaka, D. Höche, R. P. Petrauskas, C. Blawert, and M. L. Zheludkevich. A new concept for corrosion inhibition of magnesium: Suppression of iron re-deposition. *Electrochemistry Communications*, 62:5–8, 2016. https://doi.org/10.1016/j.elecom.2015.10.023.

[14] S. V. Lamaka, B. Vaghefinazari, D. Mei, R. P. Petrauskas, D. Höche, and M. L. Zheludkevich. Comprehensive screening of Mg corrosion inhibitors. *Corrosion Science*, 128:224–240, 2017. 10.1016/j.corsci.2017.07.011.

[15] T. Würger, C. Feiler, F. Musil, G. B. Vonbun-Feldbauer, D. Höche, S. V. Lamaka, M. L. Zheludkevich, and R. H. Meißner. Data Science Based Mg Corrosion Engineering. *Frontiers in Materials*, 6, 2019. 10.3389/fmats.2019.00053.

[16] D. A. Erlanson, S. W. Fesik, R. E. Hubbard, W. Jahnke, and H. Jhoti. Twenty years on: the impact of fragments on drug discovery. *Nature Reviews Drug Discovery*, 15(9):605–619, September 2016. 10.1038/nrd.2016.109.

[17] T. L. P. Galvão, G. Novell-Leruth, A. Kuznetsova, J. Tedim, and J. R. B. Gomes. Elucidating Structure-Property Relationships in Aluminum Alloy Corrosion Inhibitors by Machine Learning. *The Journal of Physical Chemistry C*, 124(10):5624–5635, March 2020. 10.1021/acs.jpcc.9b09538.

[18] C. Feiler, D. Mei, B. Vaghefinazari, T. Würger, R. H. Meißner, B. J. C. Luthringer-Feyerabend, D. A. Winkler, M. L. Zheludkevich, and S. V. Lamaka. In silico screening of modulators of magnesium dissolution. *Corrosion Science*, 163:108245, February 2020. 10.1016/j.corsci.2019.108245.

[19] T. Würger, L. Wang, D. Snihirova, M. Deng, S. V. Lamaka, D. A. Winkler, D. Höche, M. L. Zheludkevich, R. H. Meißner, and C. Feiler. Data-driven selection of electrolyte additives for aqueous magnesium batteries. *Journal of Materials Chemistry A: Materials for Energy and Sustainability*, 10:21672–21682, 2022. 10.1039/D2TA04538A.

[20] D. A. Winkler, M. Breedon, A. E. Hughes, F. R. Burden, A. S. Barnard, T. G. Harvey, and I. Cole. Towards chromate-free corrosion inhibitors: structure-property models for organic alternatives. *Green Chemistry*, 16:3349–3357, 2014. 10.1039/C3GC42540A.

[21] F. F. Chen, M. Breedon, P. White, C. Chu, D. Mallick, S. Thomas, E. Sapper, and I. Cole. Correlation between molecular features and electrochemical properties using an artificial neural network. *Materials & Design*, 112:410–418, 2016. 10.1016/j.matdes.2016.09.084.

[22] T. Würger, D. Mei, B. Vaghefinazari, D. A. Winkler, S. V. Lamaka, M. L. Zheludkevich, R. H. Meißner, and C. Feiler. Exploring structure-property relationships in magnesium dissolution modulators. *npj Materials Degradation*, 5(1):2, January 2021. 10.1038/s41529-020-00148-z.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 10.1109/CVPR.2009.5206848.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*, 60(6):84–90, May 2017. 10.1145/3065386.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 10.1109/CVPR.2016.90.

[26] C.-j. Kim, M.-j. Lee, K.-h. Hwang, and Y.-g. Ha. End-to-end deep learning-based autonomous driving control for high-speed environment. *The Journal of Supercomputing*, 78(2):1961–1982, February 2022. 10.1007/s11227-021-03929-8.

[27] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542 (7639):115–118, February 2017. 10.1038/nature21056.

[28] A. M. Smak Gregoor, T. E. Sangers, L. J. Bakker, L. Hollestein, C. A. Uyl-de Groot, T. Nijsten, and M. Wakkee. An artificial intelligence based app for skin cancer detection evaluated in a population based setting. *npj Digital Medicine*, 6(1):90, May 2023. 10.1038/s41746-023-00831-w.

[29] T. Elsken, J. H. Metzen, and F. Hutter. Neural Architecture Search. In Hutter et al. [52], pages 69–86. 10.1007/978-3-030-05318-5.

[30] T. Elsken, J. H. Metzen, and F. Hutter. Neural Architecture Search: A Survey. *Journal of Machine Learning Research (JMLR)*, 20(1):1997–2017, January 2019. http://jmlr.org/papers/v20/18-598.html.

[31] P. Ren, Y. Xiao, X. Chang, P.-y. Huang, Z. Li, X. Chen, and X. Wang. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Computing Surveys*, 54(4), May 2021. 10.1145/3447582.

[32] V. Gudivada, A. Apon, and J. Ding. Data Quality Considerations for Big Data and Machine Learning: Going Beyond Data Cleaning and Transformations. *International Journal on Advances in Software*, 10(1):1–20, 2017. https://www.iariajournals.org/software/tocv10n12.html.

[33] M. Nasser and U. K. Yusof. Deep Learning Based Methods for Breast Cancer Diagnosis: A Systematic Review and Future Direction. *Diagnostics*, 13(1), 2023. 10.3390/diagnostics13010161.

[34] A. Varma, A. Sarma, S. Doshi, and R. Nair. House Price Prediction Using Machine Learning and Neural Networks. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1936–1939, 2018. 10.1109/ICICCT.2018.8473231.

[35] V. M. V. Herrera, R. Martell-Dubois, W. Soon, G. Velasco Herrera, S. Cerdeira-Estrada, E. Zúñiga, and L. Rosique-de la Cruz. Predicting Atlantic Hurricanes Using Machine Learning. *Atmosphere*, 13(5), 2022. 10.3390/atmos13050707.

[36] D. Subramanian, R. Greiner, and J. Pearl. The relevance of relevance. *Artificial Intelligence*, 97(1):1–5, 1997. https://doi.org/10.1016/S0004-3702(97)00075-1. Relevance.

[37] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2201–2206, New York, NY, USA, 2016. Association for Computing Machinery. 10.1145/2882903.2912574.

[38] C. Khosla and B. S. Saini. Enhancing Performance of Deep Learning Models with different Data Augmentation Techniques: A Survey. In *2020 International Conference on Intelligent Engineering and Management (ICIEM)*, pages 79–85, 2020. 10.1109/ICIEM48762.2020.9160048.

[39] R. Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1984.

[40] P. Cunningham, B. Kathirgamanathan, and S. J. Delany. Feature Selection Tutorial with Python Examples. *CoRR*, 2021. 10.48550/arXiv.2106.06437.

[41] M. Kuhn and K. Johnson. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. Chapman & Hall/CRC Data Science Series. CRC Press, Taylor & Francis Group Boca Raton, 2020. https://bookdown.org/max/FES/.

[42] S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3):1–67, 2011. 10.18637/jss.v045.i03.

[43] C. T. Lynch. *Handbook of Materials Science: Volume 1 General Properties*. Routledge Revivals. CRC Press, 2020.

[44] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. 10.1002/aic.690370209.

[45] E. Oja. Data Compression, Feature Extraction, and Autoassociation in Feedforward Neural Networks. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 737–745. Elsevier Science Publishers B.V., North-Holland, 1991.

[46] D. DeMers and G. Cottrell. Non-Linear Dimensionality Reduction. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 580–587. Morgan-Kaufmann, 1992. https://proceedings.neurips.cc/paper_files/paper/1992/file/cdc0d6e63aa8e41c89689f54970bb35f-Paper.pdf.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986. 10.1038/323533a0.

[48] J. Almotiri, K. Elleithy, and A. Elleithy. Comparison of autoencoder and Principal Component Analysis followed by neural network for e-learning using handwritten recognition. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–5, 2017. 10.1109/LISAT.2017.8001963.

[49] L. S. Shapley. *A Value for n-Person Games*, In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, chapter 17, pages 307–318. Princeton University Press, Princeton, 1953. doi:10.1515/9781400881970-018.

[50] S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[51] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. 10.1109/4235.585893.

[52] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019. 10.1007/978-3-030-05318-5.

[53] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[54] L. Hernando, A. Mendiburu, and J. A. Lozano. Hill-Climbing Algorithm: Let's Go for a Walk Before Finding the Optimum. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7, 2018. 10.1109/CEC.2018.8477836.

[55] K. S. Tang, K. F. Man, S. Kwong, and Q. He. Genetic algorithms and their applications. *IEEE Signal Processing Magazine*, 13(6):22–37, 1996. 10.1109/79.543973.

[56] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983. 10.1126/science.220.4598.671.

[57] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. O'Reilly Media, Inc., 2019.

[58] B. Alberts, R. Heald, A. Johnson, D. Morgan, K. Roberts, P. Walter, J. Wilson, and T. Hunt. *Molecular Biology of the Cell*. W. W. Norton & Company, seventh edition, 2022. https://wwnorton.com/books/9780393884821.

[59] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. 10.1037/h0042519.

[60] A. LeNail. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*, 4(33):747, 2019. 10.21105/joss.00747.

[61] A. G. Ivakhnenko and V. G. Lapa. *Cybernetic predicting devices*. TR-EE ; 66-5. Purdue University School of Electrical Engineering, Lafayette, Ind, 1966.

[62] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989. 10.1162/neco.1989.1.4.541.

[63] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 10.1109/5.726791.

[64] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. 10.1007/978-3-319-24574-4_28.

[65] L. Sifre and S. Mallat. Rigid-Motion Scattering for Texture Classification. *arXiv*, 2014. 10.48550/arXiv.1403.1687.

[66] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv*, 2017. 10.48550/arXiv.1704.04861.

[67] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[68] B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv preprint, 2015. 10.48550/arXiv.1505.00853.

[69] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *International Conference on Learning Representations*, 2016. 10.48550/arXiv.1511.07289.

[70] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. 10.5281/zenodo.4724125. Software available from tensorflow.org.

[71] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[72] J. T. H. Smith, A. Warrington, and S. Linderman. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations*, 2023. https://openreview.net/forum?id=Ai8Hw3AXqks.

[73] H. Wang, J. Peng, F. Huang, J. Wang, J. Chen, and Y. Xiao. MICN: Multi-scale Local and Global Context Modeling for Long-term Series Forecasting. In *International Conference on Learning Representations*, 2023. https://openreview.net/forum?id=zt53IDUR1U.

[74] M. Ricci, N. Moriel, Z. Piran, and M. Nitzan. Phase2vec: dynamical systems embedding with a physics-informed convolutional network. In *International Conference on Learning Representations*, 2023. https://openreview.net/forum?id=z9C5dGip90.

[75] M. Ranzato, C. Poultney, S. Chopra, and Y. Cun. Efficient Learning of Sparse Representations with an Energy-Based Model. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. https://proceedings.neurips.cc/paper_files/paper/2006/file/87f4d79e36d68c3031ccf6c55e9bbd39-Paper.pdf.

[76] J. Weng, N. Ahuja, and T. S. Huang. Cresceptron: a self-organizing neural network which grows adaptively. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 576–581, 1992. 10.1109/IJCNN.1992.287150.

[77] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In F. Bach and D. Blei, editors, *Proceedings of*

the 32nd International Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, July 2015. PMLR. https://proceedings.mlr.press/v37/ioffe15.html.

[78] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization? In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. https://proceedings.neurips.cc/paper_files/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.

[79] E. J. Schiessler, T. Würger, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler, and R. C. Aydin. Predicting the inhibition efficiencies of magnesium dissolution modulators using sparse machine learning models. *npj Computational Materials*, 7(1):193, December 2021. 10.1038/s41524-021-00658-7.

[80] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016. 10.1016/j.neucom.2015.08.104. RoLoD: Robust Local Descriptors for Computer Vision 2014.

[81] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery. 10.1145/1390156.1390294.

[82] M. Sakurada and T. Yairi. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, pages 4–11, New York, NY, USA, 2014. Association for Computing Machinery. 10.1145/2689746.2689747.

[83] M. Tenorio and W.-T. Lee. Self Organizing Neural Networks for the Identification Problem. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. https://proceedings.neurips.cc/paper_files/paper/1988/file/f2217062e9a397a1dca429e7d70bc6ca-Paper.pdf.

[84] J. Moody. Fast Learning in Multi-Resolution Hierarchies. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. https://proceedings.neurips.cc/paper_files/paper/1988/file/82161242827b703e6acf9c726942a1e4-Paper.pdf.

[85] B. Zoph and Q. Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, 2017. https://openreview.net/forum?id=r1Ue8Hcxg.

[86] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019. https://openreview.net/forum?id=S1eYHoC5FX.

[87] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[88] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient Architecture Search by Network Transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. 10.1609/aaai.v32i1.11709.

[89] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient Neural Architecture Search via Parameters Sharing. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, July 2018. https://proceedings.mlr.press/v80/pham18a.html.

[90] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P.-J. Kindermans, and Q. V. Le. Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 10.1109/CVPR42600.2020.01433.

[91] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang, and Q. Tian. Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap. *ACM Computing Surveys*, 54(9), October 2021. 10.1145/3473330.

[92] E. J. Schiessler, R. C. Aydin, K. Linka, and C. J. Cyron. Neural network surgery: Combining training with topology optimization. *Neural Networks*, 144:384–393, 2021. 10.1016/j.neunet.2021.08.034.

[93] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations*, 2020. https://openreview.net/forum?id=BJlS634tPr.

[94] J. Mun, S. Ha, and J. Lee. DE-DARTS: Neural architecture search with dynamic exploration. *ICT Express*, 9(3):379–384, 2023. https://doi.org/10.1016/j.icte.2022.04.005.

[95] X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1294–1303, 2019. 10.1109/ICCV.2019.00138.

[96] Y. Xu, L. Xie, W. Dai, X. Zhang, X. Chen, G.-J. Qi, H. Xiong, and Q. Tian. Partially-Connected Neural Architecture Search for Reduced Computational Redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2953–2970, 2021. 10.1109/TPAMI.2021.3059510.

[97] X. Zhang, P. Hou, X. Zhang, and J. Sun. Neural Architecture Search with Random Labels. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10902–10911, 2021. 10.1109/CVPR46437.2021.01076.

[98] W. Wang, X. Zhang, H. Cui, H. Yin, and Y. Zhang. FP-DARTS: Fast parallel differentiable neural architecture search for image classification. *Pattern Recognition*, 136: 109193, 2023. 10.1016/j.patcog.2022.109193.

[99] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan. A Survey on Evolutionary Construction of Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 25(5):894–912, 2021. 10.1109/TEVC.2021.3079985.

[100] Y. Xue, Y. Wang, J. Liang, and A. Slowik. A Self-Adaptive Mutation Neural Architecture Search Algorithm Based on Blocks. *IEEE Computational Intelligence Magazine*, 16(3): 67–78, 2021. 10.1109/MCI.2021.3084435.

[101] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing Neural Networks Using Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[102] R. Reed. Pruning algorithms – a survey. *IEEE Transactions on Neural Networks*, 4(5): 740–747, 1993. 10.1109/72.248452.

[103] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the State of Neural Network Pruning? In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020. https://proceedings.mlsys.org/paper/2020/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf.

[104] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. 10.1016/j.neucom.2021.07.045.

[105] J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*, 2019. https://openreview.net/forum?id=rJl-b3RcF7.

[106] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning Efficient Convolutional Networks through Network Slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017. 10.1109/ICCV.2017.298.

[107] TURBOMOLE. *V7.4. A Development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2019 since 2007.* Available from TURBOMOLE GmbH, 2019.

[108] A. Mauri. alvaDesc: A Tool to Calculate and Analyze Molecular Descriptors and Fingerprints. In K. Roy, editor, *Ecotoxicological QSARs*, pages 801–820. Springer US, New York, NY, 2020. 10.1007/978-1-0716-0150-1_32.

[109] E. J. Schiessler, T. Würger, B. Vaghefinazari, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler, and R. C. Aydin. Searching the Chemical Space for Effective Magnesium Dissolution Modulators: A Deep Learning Approach using Sparse Features. *npj Materials Degradation*, 7:74, September 2023. 10.1038/s41529-023-00391-0.

[110] T. Chen, J. G. Ian, and J. Shlens. Net2Net: Accelerating Learning via Knowledge Transfer. In *International Conference on Learning Representations*, 2016. 10.48550/arXiv.1511.05641.

[111] J. Xue, J. Li, and G. Yifan. Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 2365–2369, 2013. https://www.isca-speech.org/archive_v0/interspeech_2013/i13_2365.html.

[112] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 1269–1277. Curran Associates, Inc., 2014. https://proceedings.neurips.cc/paper/2014/file/2afe4567e1bf64d32a5527244d104cea-Paper.pdf.

[113] R. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Los Alamitos, CA, USA, December 2015. IEEE Computer Society. 10.1109/ICCV.2015.169.

[114] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, September 1936. 10.1007/BF02288367.

[115] M. W. Shelley. *Frankenstein, or, The modern Prometheus*. Lackington, Hughes, Harding, Mavor & Jones, London, 1818.

[116] G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978. http://www.jstor.org/stable/2958889.

[117] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and N. Andrew. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[118] A. Malhotra. Dynamic Scheduling Routines in Neural Architecture Search. Master's thesis, Kiel University, 2022.

[119] R. Liu, J. Cao, P. Li, W. Sun, Y. Zhang, and Y. Wang. NFP: A No Fine-tuning Pruning Approach for Convolutional Neural Network Compression. In *2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 74–77. IEEE, 2020. 10.1109/ICAIBD49809.2020.9137429.

[120] E. J. Schiessler, R. C. Aydin, and C. J. Cyron. ECToNAS: Evolutionary Cross-Topology Neural Architecture Search. *manuscript in preparation for submission*, 2024. 10.48550/arXiv.2403.05123.

Appendix A

Peer reviewed and published articles

A.1 Publication 1

Neural network surgery: Combining training with topology optimization

E. J. Schiessler, R. C. Aydin, K. Linka and C. J. Cyron

E.J.S., R.C.A., K.L. and C.J.C. contributed to the conception and design of the study. E.J.S. wrote the supporting code, ran the computational experiments, created the figures, and wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

Reprinted from E. J. Schiessler, R. C. Aydin, K. Linka, and C. J. Cyron. Neural network surgery: Combining training with topology optimization. *Neural Networks*, 144:384–393, 2021. 10.1016/j.neunet.2021.08.034,

Neural Networks 144 (2021) 384–393

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Neural network surgery: Combining training with topology optimization

Elisabeth J. Schiessler [a,*], Roland C. Aydin [a,*], Kevin Linka [b], Christian J. Cyron [a]

[a] Helmholtz-Zentrum Hereon, Institute of Material Systems Modeling, Dept. of Machine Learning and Data, Max–Planck–Straße 1, 21502 Geesthacht, Germany
[b] Hamburg University of Technology, Institute of Continuum and Materials Mechanics, Eißendorfer Straße 42, 21073 Hamburg, Germany

ARTICLE INFO

Article history:
Received 31 March 2021
Received in revised form 27 August 2021
Accepted 30 August 2021
Available online 7 September 2021

Keywords:
Neural architecture search
Topology optimization
Singular value decomposition
Genetic algorithm

ABSTRACT

With ever increasing computational capacities, neural networks become more and more proficient at solving complex tasks. However, picking a sufficiently good network topology usually relies on expert human knowledge. Neural architecture search aims to reduce the extent of expertise that is needed. Modern architecture search techniques often rely on immense computational power, or apply trained meta-controllers for decision making. We develop a framework for a genetic algorithm that is both computationally cheap and makes decisions based on mathematical criteria rather than trained parameters. It is a hybrid approach that fuses training and topology optimization together into one process. Structural modifications that are performed include adding or removing layers of neurons, with some re-training applied to make up for any incurred change in input–output behaviour. Our ansatz is tested on several benchmark datasets with limited computational overhead compared to training only the baseline. This algorithm can achieve a significant increase in accuracy (as compared to a fully trained baseline), rescue insufficient topologies that in their current state are only able to learn to a limited extent, and dynamically reduce network size without loss in achieved accuracy. On standard ML datasets, accuracy improvements compared to baseline performance can range from 20% for well performing starting topologies to more than 40% in case of insufficient baselines, or reduce network size by almost 15%.

1. Introduction

A common problem for any given machine learning task making use of artificial neural networks (ANNs) is how to choose a sufficiently good network topology. Picking one that is too small may not yield acceptable prediction accuracy. To improve results, one can keep adding structural elements to the network until the desired accuracy value has been reached. Too large networks on the other hand may cause an explosion in computational cost for both training and evaluation. Finding the optimal balance is heavily dependent on the given task, dataset and further hyperparameters, and often requires expert domain knowledge. A priori optimization is not easily possible, since reliable estimates on network behaviour already require training results, and no generalization exists which topology will fit which problem. Researchers have applied a number of search strategies such as random search (Li & Talwalkar, 2019), Bayesian optimization (Kandasamy, Neiswanger, Schneider, Poczos, & Xing, 2018), reinforcement learning (Zoph & Le, 2017), and gradient-based methods (Dong & Yang, 2019; Li, Khodak, Balcan, & Talwalkar, 2021; Liu, Simonyan, & Yang, 2019; Wang, Cheng, Chen, Tang, & Hsieh, 2021; Xu et al., 2020). Another technique applied since at least (Miller, Todd, & Hegde, 1989) are so called (neuro-) evolutionary algorithms. These algorithms serve to evolve the network architecture, often also training network weights at the same time (Elsken, Metzen, & Hutter, 2019).

In this paper we propose a novel training regime incorporating a genetic algorithm that reduces computational cost compared to state of the art approaches of this kind (Dong & Yang, 2019; Li & Talwalkar, 2019). We achieve this by re-using network weights for competing modification candidates instead of retraining each net from scratch, branching off modification candidates during training, and letting them compete against each other until a new main branch is selected. This fuses the evolutionary optimization paradigm with the ANN training into an integrated framework that folds both processes into a single training/topology optimization hybrid. As such, evolutionary steps are not carried out by a meta-controller or other black-box-like implementations, but instead make use of mathematical tools such as singular value decomposition (SVD) and the Bayesian information criterion (BIC) (Schwarz, 1978) for network weight analysis, decision

* Equal contribution.
E-mail addresses: elisabeth.schiessler@hereon.de (E.J. Schiessler), roland.aydin@hereon.de (R.C. Aydin).

https://doi.org/10.1016/j.neunet.2021.08.034

making, and structural modifications. Network modifications are performed by adapting existing weights such as to incur minimal changes to input–output behaviour.

Our framework for a combined ANN training and neural architecture search consists of three main components: a module that can perform a number of minimally invasive network operations ("surgeries"), a module that analyses network weights and can give recommendations which modifications are most likely to increase (validation) accuracy, and finally a module that serves as a genetic algorithm (the "Surgeon"), containing the former two while gradually evolving any given starting network. With the Surgeon, we are able to evolve and improve models for several benchmark datasets and varying starting topologies. We achieve particularly good results on starting topologies that would a posteriori have proven to be suboptimal. A great benefit of our approach is that it adds topology optimization to ML training while incurring very limited additional computational costs. Convergence is reached for all test cases within a few hours.

The supporting code can be accessed via https://github.com/ElisabethJS/neural-network-surgery.

This paper contributes a computationally cheap ansatz for a genetic neural architecture search algorithm that makes evolutionary decisions based on mathematical analysis.

2. Related work

Neural architecture search (NAS) has been an increasingly popular research topic for many years (Elsken et al., 2019), starting as early as Miller et al. (1989), who presented one of the earliest neuro-evolutionary algorithms to search for suitable network topologies. Recent approaches by Dong and Yang (2019), Li and Talwalkar (2019), and Zoph and Le (2017) reach competitive performance on benchmark datasets such as CIFAR-10. However, this often comes at the cost of vast computational resources, with Zoph and Le (2017) making use of up to 800 GPUs for several weeks.

Cai, Chen, Zhang, Yu, and Wang (2018) attempt to reduce computational costs by re-using network weights, as well as training and applying a reinforcement meta-controller for structural decisions. They make use of a number of function-preserving transformations (net2net) introduced by Chen, Goodfellow, and Shlens (2016), and extend them to allow also non-sequential network structures, such as DenseNet (Huang, Liu, van der Maaten, & Weinberger, 2017). DiMattina and Zhang (2010) introduce and rigorously prove conditions, under which gradual changes of the parametrization of a neural network are possible, while keeping the input–output behaviour constant.

İrsoy and Alpaydın (2020) learn the network structure via so-called "budding perceptrons", in which an extra parameter is learned for each layer, that indicates whether or not any given node needs to branch out again or be removed altogether. Their method focuses on growing the network to the required size from a minimal starting topology. Frankle and Carbin (2019) present a method to identify particularly good network initializations that can train sparse networks to competitive accuracy. Another approach in NAS is to prune down from a larger starting topology (Blalock, Gonzalez Ortiz, Frankle, & Guttag, 2020). Popular pruning techniques include applying SVD to existing network weights (Denton, Zaremba, Bruna, LeCun, & Fergus, 2014; Girshick, 2015; Xue, Li, & Gong, 2013).

There are also a number of neural architecture search strategies that do not depend on manual network modifications. Liu et al. (2019) introduced DARTS, a method for differential architecture search that re-formulates the task of searching for network architectures as a graph optimization problem, where all possible network configurations are represented as nodes on a directed acyclic graph. This technique has rapidly become very popular and has seen a great number of extensions in various directions, such as Dong and Yang (2019), Li et al. (2021), Wang et al. (2021) and Xu et al. (2020). The downside of the DARTS based algorithms is that all possible network variations are predefined in advance and cannot be adapted during training based on the network state. Additionally, since all nodes of a certain hierarchy level need to be interchangeable or even skippable, the connections between network blocks are highly restricted with regard to network structure.

The novelty of our research lies in combining existing tools such as net2net (Chen et al., 2016) and SVD with a genetic algorithm that modifies the given network in a decision based process instead of utilizing a black-box like decision module, while retaining a very high level of structural freedom. We repeatedly generate functionally equivalent but structurally different networks that are then trained for a short number of epochs, after which their performance is compared and the best candidates are retained. Ultimately this yields a fully trained neural network with an optimized topology. To our best knowledge no such method has yet been proposed.

3. Methods

This work introduces and utilizes three main modules:

- modification module: performs network modifications ("surgeries") so as to incur minimal changes to input–output behaviour.
- recommendation module: analyses network weights and gives recommendations on which operations are most likely to improve network accuracy.
- "the Surgeon": a genetic algorithm that links the above two modules, and gradually evolves a given starting network.

3.1. The modification module

We want to be able to carry out a number of different modifications that can restructure network architecture while keeping the input–output behaviour intact, and in particular avoid loss of prediction quality. In cases where this is not possible we aim for minimal impact changes instead. This yields network variations that are structurally different but functionally equivalent.

In mathematical terms, let the output of a dense layer be given by

$$f(x) = \Phi(\mathcal{A} \cdot x + b), \tag{1}$$

with activation function Φ, weight matrix $\mathcal{A}$ and bias vector b, as well as an arbitrary input x to that layer. We perform four different types of modifications, namely adding or removing neurons or whole layers. In particular, we are looking for $\tilde{f}(x)$ such that

$$\tilde{f}(x) \equiv f(x)\ \forall x, \text{ but } \tilde{f} \neq f. \tag{2}$$

Activation functions used in neural networks are usually nonlinear, or piecewise linear at best. Thus changing the activation function will in general not produce equal results for arbitrary input values. Finding modifications that still suffice Eq. (2) is therefore synonymous to adequately adapting the affected network weight matrices and bias vectors.

Adding layers. Under (at least) piecewise linear activation functions such as ReLu, one can always add neurons to a hidden layer, or even add whole layers, without any change to the overall network behaviour (Chen et al., 2016). Adding a whole layer is done by using an identity matrix as a new weight matrix for this inserted layer. In particular, the added layer will have the same

number n of neurons as the following layer. We initialize the weights as an identity matrix $\mathcal{I} \in \mathbb{R}^n$. Let Φ denote the activation function of a dense layer and x an arbitrary input, then

$$\Phi(x) = \Phi(\mathcal{I}\Phi(x)) \tag{3}$$

if and only if Φ is at least piecewise linear. In particular, for ReLu activation, adding layers without changing the input–output behaviour is possible.

Adding neurons. For adding neurons to an existing layer, consider the following example. Let $x \in \mathbb{R}^n$ be the input to a layer with weights $\mathcal{A} \in \mathbb{R}^{m\times n}$, and $\mathcal{B} \in \mathbb{R}^{k\times m}$ the next layer's weights, and let the layer's bias vector be zero w.l.o.g. Assume the activation function Φ acting between the two layers to be an identity mapping. Then the output $y \in \mathbb{R}^k$ is given by

$$y = \mathcal{B} \cdot \Phi(\mathcal{A}x) = \mathcal{B} \cdot (\mathcal{A}x). \tag{4}$$

In particular, the jth entry of y is

$$y_j = (b_{j1}, b_{j2}, \ldots, b_{jm}) \cdot \begin{bmatrix} \sum_{i=1}^n x_i a_{1i} \\ \sum_{i=1}^n x_i a_{2i} \\ \vdots \\ \sum_{i=1}^n x_i a_{mi} \end{bmatrix} \tag{5}$$

$$= b_{j1} \sum_{i=1}^n x_i a_{1i} + \cdots + b_{jm} \sum_{i=1}^n x_i a_{1m}, \tag{6}$$

where a_{ij} is the ij element of $\mathcal{A}$, and b_{ij} the ij element of $\mathcal{B}$. We now arbitrarily pick unit m and duplicate its incoming and outgoing weights. Note that we also have to divide by 2 in order to keep the total sum constant.

$$y_j = b_{j1} \sum_{i=1}^n x_i a_{1i} + \cdots + \frac{b_{jm}}{2} \sum_{i=1}^n x_i a_{1m} + \frac{b_{jm}}{2} \sum_{i=1}^n x_i a_{1m} \tag{7}$$

$$= (b_{j1}, b_{j2}, \ldots, \frac{b_{jm}}{2}, \frac{b_{jm}}{2}) \cdot \begin{bmatrix} \sum_{i=1}^n x_i a_{1i} \\ \sum_{i=1}^n x_i a_{2i} \\ \vdots \\ \sum_{i=1}^n x_i a_{mi} \\ \sum_{i=1}^n x_i a_{mi} \end{bmatrix} \tag{8}$$

This methods yields the exact same output $y \in \mathbb{R}^k$ given input $x \in \mathbb{R}^n$, but the weight matrices $\mathcal{A}$ and $\mathcal{B}$ are now of dimension $(m+1, n)$ and $(k, m+1)$ respectively, and can be extended to include a non-zero bias term in a similar fashion.

Recall now that we chose the activation Φ to be the identity mapping. For any other activation function, the equation in line (7) holds true, if and only if this activation is at least piecewise linear.

Chen et al. (2016) base their Net2WiderNet and Net2DeeperNet transformation on these steps.

Removing neurons. Removing neurons from a layer is rarely possible without changing the input–output behaviour unless some of the units are degenerate to begin with, i.e. the weight matrix is of reduced rank. For a modification with minimal impact on input–output behaviour we need the closest possible projection onto a lower rank subspace. As a measure for closeness we employ the *Frobenius norm*, defined as follows.

Let $\|\cdot\|_F$ denote the *Frobenius norm*, with

$$\|\mathcal{A}\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 = \text{trace}(\mathcal{A}^T\mathcal{A}), \tag{9}$$

where $\mathcal{A} \in \mathbb{R}^{m\times n}$ is an arbitrary, real-valued matrix of rank $k \leq \min(m, n)$.

The Eckart–Young(–Mirsky) theorem states that the closest projection onto a lower rank subspace can be found by applying Singular Value Decomposition (SVD) to the weight matrix:

Let $\mathcal{A} \in \mathbb{R}^{m\times n}$ be the (weight) matrix of some layer L_j of interest in a neural network, then there exists a representation

$$\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T \tag{10}$$

with orthogonal $\mathcal{U} \in \mathbb{R}^{m\times m}$, $\mathcal{V} \in \mathbb{R}^{n\times n}$ and rectangular diagonal $\Sigma \in \mathbb{R}^{m\times n}$ with $k \leq \min(m, n)$ non-negative real entries σ_i along its diagonal. This representation is called the *Singular Value Decomposition* of $\mathcal{A}$. The σ_i are called *singular values* of $\mathcal{A}$, and are usually given in descending order, i.e. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$. It can be shown that $\|\mathcal{A}\|_F^2 = \sum \sigma_i^2(\mathcal{A})$, since $\sigma_i(\mathcal{A})$ corresponds to the square root of the ith non-zero eigenvalue of $\mathcal{A}^T\mathcal{A}$. Note that $\mathcal{U}$ and $\mathcal{V}$ are projections from the $(m \times n)$ vector space spanned by $\mathcal{A}$ to the $(k \times k)$ vector space spanned by Σ and back.

In order to reduce the rank of $\mathcal{A}$ to $r < k$, we set $\sigma_i = 0\ \forall i > r$, and drop associated columns/rows of $\mathcal{U}$ and $\mathcal{V}$, which is feasible since by definition $\sigma_i \geq \sigma_{i+1}$. We then project to a smaller matrix $\tilde{\mathcal{A}} \in \mathbb{R}^{m\times r}$ by computing $\mathcal{U}\Sigma$. In order to properly re-connect the reduced layer L_j to the following layer L_{j+1}, we have to modify L_{j+1}'s weight matrix to accept input of length r. In particular, we need to project back to the original layer size, to ensure that the layer weight shapes match again. We achieve this by multiplying from the left the weight matrix of L_{j+1} with $\mathcal{V}^T$.

Projecting onto a lower rank subspace by setting singular values to zero is sometimes called truncated SVD, and is used in network pruning (Denton et al., 2014; Girshick, 2015; Xue et al., 2013). This technique adds an intermediate layer of (potentially much fewer) neurons, which in case of very large weight matrices can drastically reduce the overall connection count.

Note that this projection method is not concerned at all with a potential activation function after the network's modified layer. As mentioned previously, changing a layer's activation function is in general not possible without changes to input/output behaviour, since activation functions are usually non-linear. We therefore do not perform any additional modifications to counterbalance a change of activation.

Removing layers. We remove whole layers in a similar fashion. Let $\mathcal{A}_j, b_j$ be the weight matrix and bias vector of some dense Layer L_j, and $\mathcal{A}_{j+1}, b_{j+1}$ those of the subsequent Layer L_{j+1}. We remove the Layer L_j by simply dropping it, and modify L_{j+1}'s weights by matrix multiplication, yielding $\tilde{L}_{j+1}$

$$\tilde{\mathcal{A}}_{j+1} = \mathcal{A}_j \cdot \mathcal{A}_{j+1}, \tag{11}$$

$$\tilde{b}_{j+1} = b_j \cdot \mathcal{A}_{j+1} + b_{j+1}. \tag{12}$$

This again ignores any activation function between the two layers, and will thus cause a change in input–output behaviour.

Recuperation from surgery. As we have seen, network modifications will in general cause a small change in input–output behaviour, typically leading to a loss in prediction accuracy. In order to make up for this, all network modifications are given a small amount of recuperative training (several batches) before any comparison is made. The retraining amount was determined by trials and statistical analysis based on a dataset[1] which we do not otherwise use in our results, to avoid overfitting the search algorithm to a specific dataset.

3.2. The recommendation module

For the decision *when* to execute *which* network modification, we perform a two-step analysis.

Analytical criterion for model selection. As a first order criterion, we compute the amount of information carried by each neuron by looking at the layer's singular values. The number of

[1] MNIST, LeCun, Bottou, Bengio, and Haffner (1998).

neurons that may be removed from a layer depends on the count of singular values that are (close to) zero, or are several orders of magnitude smaller than the layer's largest singular value:

$$n_r = \{\sigma_i,\ i = 1 \dots k \mid \sigma_i < \epsilon_1 \sigma_1 \vee \sigma_i < \epsilon_2,\ \epsilon_1,\ \epsilon_2 \in \mathbb{R}^+\} \tag{13}$$

This number is compared to the layer's total neuron count n_l. Let h be the number of hidden layers in the network, and i be the index of the layer in which the modification was performed. Then each modification candidate is given a score between 0 and 1, calculated as follows:

- add neurons: $1 - n_r/n_l$
- add layer: $(1 - n_r/n_t) \cdot i/h$
- remove neurons: n_r/n_l
- remove layer: 1, if $n_r = n_l$, otherwise $(n_r/n_l) \cdot i/h$

In particular, the higher the layer number, the more likely a layer is added or removed. For further refinement in the future, we aim to replace this selection criterion with a more sophisticated formula, which would improve the optimization process beyond the results presented herein.

Statistical criterion for model selection. The second order decision basis is derived from the Bayesian information criterion (BIC), also known as Schwarz information criterion. It was derived by Schwarz (1978) to address the problem of selecting between (statistical) models of different dimension. It takes into account the number of parameters of the given model, the sample size of the input data, as well as the a-posteriori model error computed from a likelihood function of the model given its parameters and input data. Since methods from Bayesian statistics are applied, it is assumed that the underlying data are independent and identically distributed from a family of allowed distributions.

The BIC is given by

$$\mathrm{BIC} = \ln n \cdot k - 2 \cdot \ln L, \tag{14}$$

where k is the number of model parameters, n the sample size, and L the likelihood function.

In our application, sample size is constant throughout the whole process. L needs to be estimated or calculated a-posteriori after a network operation has been performed. Our modifications are intended to keep the change in input–output behaviour minimal, therefore the difference in L will be very small between any two modifications. Eq. (14) thus becomes

$$\mathrm{BIC} \approx c_1 \cdot k + c_2 + \Delta L, \tag{15}$$

where c_1 and c_2 are constant and ΔL is the (small) change in error depending on the performed operation. Since the number of parameters k may be well above 10^6 and $c_1 = \ln n \gg 1$ is dependent on the sample size, we neglect ΔL and the constants, and directly use k as a second order constraint when deciding which modifications to apply.

Thus, all potential network modifications are ranked by two parameters. The Surgeon has two modes: it can either pick the top n ranking operations per decision step (with n being a tunable hyperparameter), or select the highest ranking operation of each type. Our experiments are performed with the latter option.

3.3. "The Surgeon": A genetic algorithm for neural architecture search

The final module is the Surgeon, the genetic algorithm that searches for an optimized network architecture while training network weights at the same time. We use the term genetic algorithm to describe a searching meta heuristic inspired by biological processes such as evolution, mutation, and selection. In this first paper about our new ansatz, we limit our focus on perhaps the most ubiquitous type of architecture: sequential networks (i.e. without recurrent or skip connections) consisting only of fully connected layers. As Cai et al. (2018) have shown, however, a number of the above described tools can be generalized easily also to non-sequential networks, as well as convolutional layers.

The rough idea behind the Surgeon is to alternate training and network optimization phases. Several competing topologies, called *branches*, may be retained concurrently. During the optimization phase, modification candidates are created for each such branch. From these, the n best performing ones are kept and put through another training step. One such training and optimization cycle is depicted in Fig. 1.

Algorithm 1 The Surgeon

```
function PerformSurgery(m)
    input: model m
    output: optimized network m_opt

    initial modification m_0 ← initialized from model m
    train m_0 for initial number of epochs
    list of current branches B ← initialized with modif. m_0
    while termination criteria not met do      ▷ e.g. max. epochs
        list of potential modification candidates M_cand ← ∅
        list of chosen modification candidates M_sel ← ∅
        for each branch in current branches B do
            determine possible modification candidates
            add candidates to M_cand
        repeat
            M_sel ← chosen from M_cand          ▷ e.g. 7 competitors
            re-train M_sel for small no. of batches ▷ e.g. 15 batches
            compare M_sel
            keep n top scoring as new current branches
            update current branches B           ▷ e.g. best 2 branches
            train B to full epoch step          ▷ e.g. 10 epochs
        until improvement achieved or max re-tries reached
    select final best scoring branch m_opt from B
    return fully trained optimized network m_opt
```

The overall structure of the Surgeon can be seen in algorithm 1. First, the provided model is pre-trained for an initial number of epochs, then the list of current branches is initialized with it. The choice of how many branches at most are being kept concurrently is a hyperparameter setting, and has a great influence on the total computational cost of the Surgeon.

We then evolve and continuously update the list of concurrent branches until termination criteria, such as the maximal number of epochs (if limited by computational resources) or a minimum accuracy threshold (if the topology optimization is used without a computational resource bottleneck), are met. At each decision point, the recommendation module analyses all networks in the list of current branches, and ranks all potential modifications. From these, in line 15 of algorithm 1, we select the most promising candidates, and perform the selected operations using the modification module.

The generated candidates are re-trained for several batches, to make up for lost performance. Our main objective function is to maximize the accuracy achieved by any given neural network through continuous network surgery. It is therefore sensible to retain those network candidates in algorithm 1, line 19 that reach the highest validation accuracy score. Note that we are scoring on validation accuracy instead of accuracy to avoid overfitting.

Focusing on accuracy alone is a greedy approach and carries the risk of getting stuck in local optima. We overcome this by additionally rewarding modification candidates that show a greater

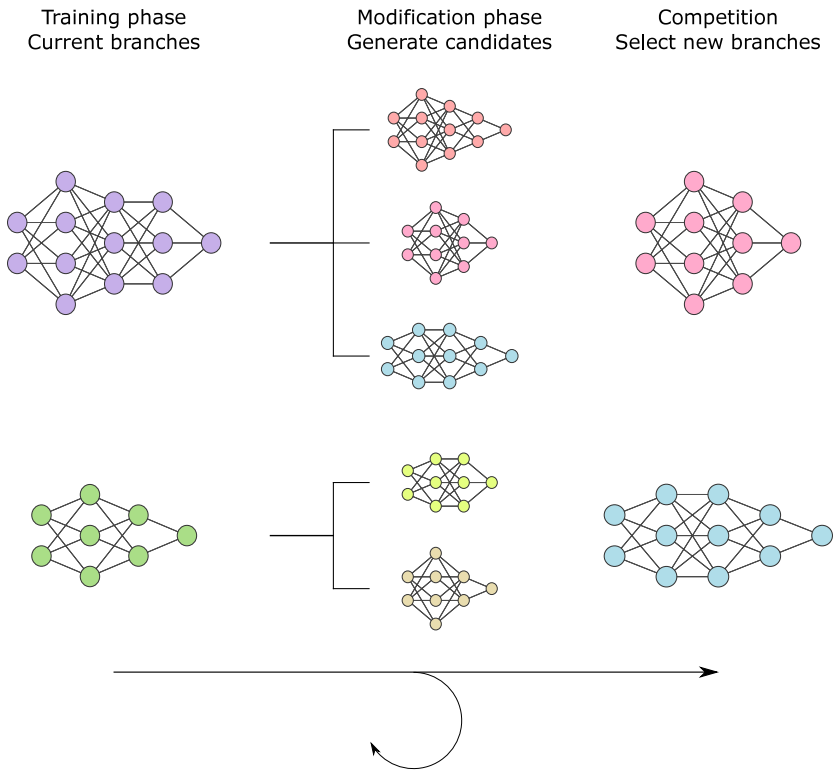


Fig. 1. Graphical representation of the Surgeon. Each cycle starts with a training phase, where all current branches are trained for a fixed number of epochs. Then the modification module generates new candidates for each branch based on the results provided by the analysis module. These candidates receive an appropriate amount of recuperative training, after which the winners are selected out of the set of all candidates, thus forming the new current branches. Network graphs created using LeNail (2019).

accuracy gain. However we need to make a distinction when rewarding this gain. We share Cai et al. (2018)'s rationale that an increase of 1% needs to be weighed higher in case it happens from 90 to 91% accuracy rather than 70 to 71%. At the same time, if an operation keeps the accuracy constant at e.g. 95% we can assume that a local optimum may have been reached. Therefore an operation that leads to an accuracy increase from 90% to 94%, showing potential for further improvement, should be regarded higher even though the reached accuracy is lower. Lastly we do not wish to neglect network size. A 1% accuracy increase might never be favourable at all, if the required increase in network size is "too big".

Note that it is hard to define when a network has indeed become "too big", a fact that is emphasized by the large number of publications dealing with pruning techniques (Blalock et al., 2020).

We need to find a way to balance these three components (accuracy, accuracy gain, network size), and to create a composite score by which we can rank the performance of current branches. As an additional restriction, the composite score should not depend on any global candidate statistics, nor do we want to set a global limit for network size. Therefore we cannot in any meaningful way regard the total number of parameters of a modification candidate, since we lack overall comparison. Instead, for each candidate, we store the network size as a fraction of the candidate's parent's network size. Thus, a network size fraction greater than 1 indicates growth, a fraction smaller than 1 indicates shrinking, and the identity operation yields a size fraction of exactly 1.

We want the scoring function as well as its first order derivative to be strictly increasing with accuracy gain, but decreasing with size fraction. This behaviour needs to hold even when accuracy gain becomes 0 or network size fraction becomes 1. The following scoring function fulfils all specified requirements:

$$s = a + \frac{\exp(\Delta a)}{\exp(\Delta f)} \tag{16}$$

where a is the current accuracy, Δa the current accuracy gain, and Δf the current network size fraction, and is adapted from Cai et al. (2018). A scoring example can be seen in Table 1.

Table 1
Scoring example. **A** denotes accuracy, **AG** accuracy gain, **PF** parameter fraction and **S** the calculated score. The layer number is given in reference to the hidden layers. Winning operations, that are kept as new concurrent branches, are indicated with an asterisk.

Operation	Position	A	AG	PF	S
Remove layer*	Layer 5	0.6023	0.0274	0.9977	0.9812
Identity*	–	0.6016	0.0268	1.0000	0.9795
Remove 1 Neuron	Layer 5	0.5829	0.0081	0.9997	0.9539
Add layer	Before layer 4	0.5773	0.0025	1.0030	0.9450
Add 20 Neurons	Layer 2	0.5678	−0.0070	2.6544	0.6377

As an option to alleviate greediness, the Surgeon can keep a one cycle memory. In this case, the newly selected concurrent branches are compared to previously best scoring ones, and retained only if their achieved accuracy is at least as good as the previous value. Should this not be the case, they are discarded and we backtrack one step. New potential candidates are provided by the recommendation module, and we train and compare these. Finally the best scoring branch is returned as the optimized network.

Table 2
Starting topologies for the Surgeon. Each topology additionally has a reshape layer as its input layer, as well as a dense layer without activation function as its output layer. The neuron count in the input and output layers is dependent on the dataset.

Name	Hidden layer count	Hidden layer sizes	Activation
Small	1	10	ReLu
Medium	3	10 - 10 - 10	ReLu
Large	3	300 - 100 - 100	ReLu

4. Experiments

4.1. Data

We evaluate the performance of the Surgeon on several standard benchmark datasets (SVHN, CIFAR-10, CIFAR-100, EuroSAT, EMNIST, Fashion-MNIST), which are described below. The SVHN dataset was downloaded manually from http://ufldl.stanford.edu/housenumbers/. The CIFAR-10 dataset was fetched from the `keras.datasets` catalogue. All others are fetched from the `tensorflow-datasets` catalogue, batched, and shuffled. As an additional preprocessing step, we normalize the data to be within the range [0, 1].

We pick three starting topologies that are described in Table 2, and perform several runs of the Surgeon on each one. Note that Table 2 omits the input layer, which is always a reshape layer that ensures the input data is formatted as a 1-dimensional vector instead of a multi-dimensional array, as well as the final dense layer. The Surgeon never adapts these two layers, as they are inherently determined by the dataset through the shape of the training data and number of output classes.

The small starting topology is consciously chosen to be insufficient for most of the regarded datasets, to mimic a case where a model is unknowingly trained with an inadequate network architecture. We average our results over several runs and several random seeds, and compare to results achieved by simply training the starting topology for the same number of epochs. Detailed machine properties and hyperparameter settings are listed in Appendix.

The SVHN dataset. The (Google) Street View House Number (SVHN) dataset was published by Netzer et al. (2011). It contains 73,257 training, as well as 26,032 validation colour images. We use the cropped version, where images are of size 32×32 pixels, and fall into 10 classes according to the numbers 0–9. Additionally, 531,131 extra images of lower difficulty are available but not currently used.

The CIFAR-10 dataset. The Canadian Institute For Advanced Research (CIFAR)-10 dataset was published by Krizhevsky (2009). It contains 60,000 32×32 pixel colour images that are evenly divided into 10 classes. 10,000 of the images are set aside for validation purposes.

The CIFAR-100 dataset. The CIFAR-100 dataset is equivalent to the CIFAR-10 dataset, except that samples are evenly divided into 100 classes.

The EuroSAT dataset. The EuroSAT dataset was published by Helber, Bischke, Dengel, and Borth (2018, 2019). It is based on Sentinel-2 satellite images consisting of 27000 labelled and geo-referenced colour images of size 64×64 pixels, that belong to 10 classes. We make use of the RGB version that contains only the optical red, green and blue frequency bands.

The EMNIST dataset. The EMNIST dataset was published by Cohen, Afshar, Tapson, and van Schaik (2017). It contains greyscale handwritten character digits of size 28×28 pixels that are derived from the NIST special database. They depict the numbers from 0–9 and thus fall into 10 classes. 697,932 training as well as 116,323 validation samples are provided.

Table 3
Statistics over all baseline and Surgeon runs. We report means and standard deviations.

Topology	Small	Medium	Large
SVHN			
Baseline accuracy	0.20 ± 0.00	0.34 ± 0.12	0.78 ± 0.01
Surgeon accuracy	0.29 ± 0.16	0.58 ± 0.04	0.79 ± 0.01
Rel. accuracy incr. [%]	45.00	70.59	1.28
Param. fraction incr. [%]	16.48 ± 52.0	1.97 ± 7.13	−7.23 ± 32.48
CIFAR-10			
Baseline accuracy	0.28 ± 0.01	0.32 ± 0.01	0.49 ± 0.00
Surgeon accuracy	0.34 ± 0.01	0.38 ± 0.06	0.51 ± 0.00
Rel. accuracy incr. [%]	21.43	18.75	4.08
Param. fraction incr. [%]	1.12 ± 2.85	203.87 ± 417.45	2.12 ± 3.13
CIFAR-100			
Baseline accuracy	0.12 ± 0.01	0.12 ± 0.01	0.20 ± 0.00
Surgeon accuracy	0.19 ± 0.00	0.14 ± 0.01	0.20 ± 0.00
Rel. accuracy incr. [%]	58.29	17.74	2.89
Param. fraction incr. [%]	865.44 ± 0.00	−0.18 ± 11.94	7.40 ± 27.69
EuroSAT			
Baseline accuracy	0.11 ± 0.00	0.24 ± 0.10	0.60 ± 0.02
Surgeon accuracy	0.54 ± 0.05	0.57 ± 0.03	0.66 ± 0.01
Rel. accuracy incr. [%]	383.09	215.00	10.37
Param. fraction incr. [%]	−14.31 ± 10.68	−12.87 ± 13.83	3.24 ± 4.15
EMNIST			
Baseline accuracy	0.67 ± 0.00	0.63 ± 0.01	0.83 ± 0.00
Surgeon accuracy	0.73 ± 0.00	0.71 ± 0.01	0.85 ± 0.00
Rel. accuracy incr. [%]	8.13	12.10	1.54
Param. fraction incr. [%]	470.44 ± 0.00	16.98 ± 47.30	13.02 ± 9.78
Fashion-MNIST			
Baseline accuracy	0.85 ± 0.01	0.85 ± 0.01	0.89 ± 0.00
Surgeon accuracy	0.85 ± 0.00	0.85 ± 0.00	0.89 ± 0.00
Rel. accuracy incr. [%]	0.47	0.65	0.06
Param. fraction incr. [%]	−0.36 ± 10.44	−3.83 ± 11.64	24.94 ± 86.73

The Fashion-MNIST dataset. The Fashion-MNIST dataset was published by Xiao, Rasul, and Vollgraf (2017). It contains greyscale fashion image data taken from Zalando's[2] article catalogue that fall into 10 categories. The dataset consists of 60,000 training as well as 10,000 validation samples of size 28×28 pixels.

4.2. Results

We apply the Surgeon to each combination of the above datasets and starting topologies (cf. Table 2). We do so a total of 15 times, re-initializing the `numpy` and `tensorflow` modules with a new random seed after every 3 runs, and report average statistics (cf. Table 3). As a baseline, we train the starting topology for the same number of epochs with each random seed. Note that throughout the entire section, unless otherwise stated, we report achieved validation accuracies rather than training accuracies.

To avoid overfitting on any specific dataset, we fix some hyperparameters for training (such as batch size, optimizer, and learning rate) before starting any runs with the Surgeon (cf. Appendix). No additional fine tuning is performed on any model.

Overall Surgeon performance. Table 3 and Fig. 2 both show an overview of average Surgeon performance for all starting topologies and datasets. We can see that in all cases, the Surgeon reaches or outperforms the result of the baseline with regard to validation accuracy. We are able to observe two types of behaviour. In cases where the baseline accuracy is already high to begin with, the relative accuracy increase achieved by the Surgeon is comparatively low. However the Surgeon is often able

[2] A fashion company specialized in online commerce.

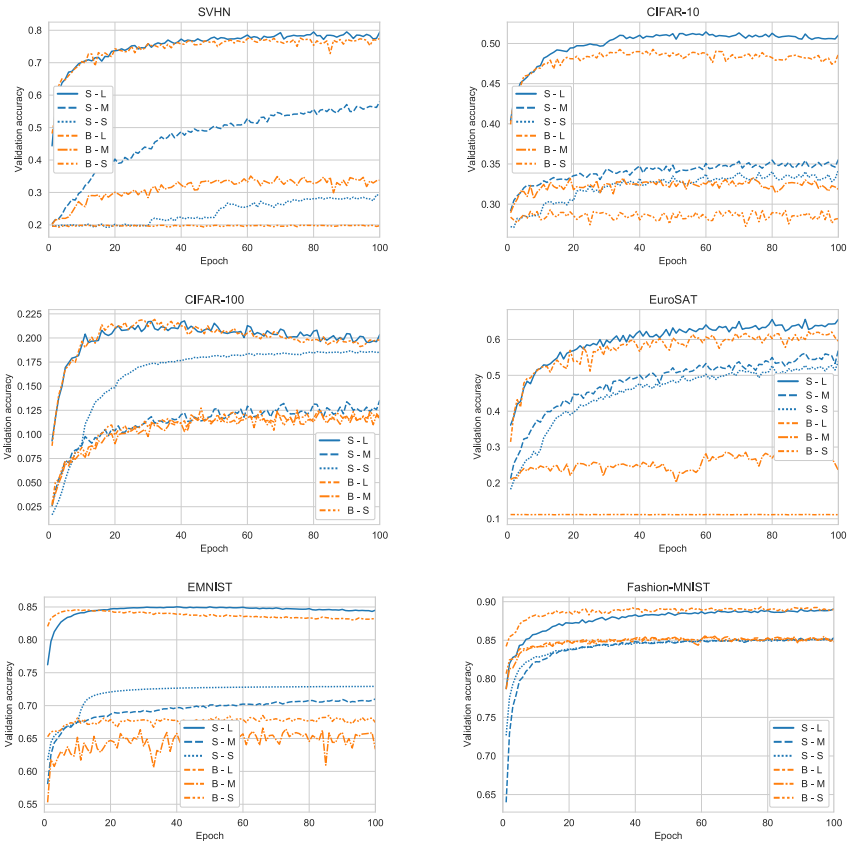


Fig. 2. Average performances of the Surgeon (S) compared to the baseline (B) for all three starting topologies (L — large, M — medium, S — small), depicted per dataset.

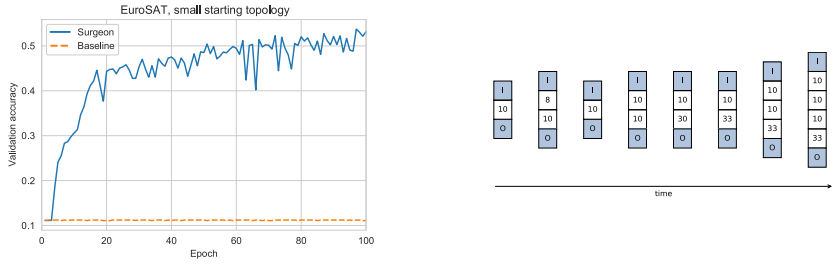


Fig. 3. Left: Single run of the Surgeon on the EuroSAT dataset and small starting topology. Right: Evolution of the applied topology during the run. Boxes with marked the letters I and O designate the input and output layers, the numbers in the remaining boxes indicate the number of neurons in the respective hidden layers.

to decrease the required amount of network parameters without loss in accuracy. On the other hand we are able to observe quite significant improvements in accuracy in cases where the starting topology is sub optimal. In fact, as can be seen in Fig. 2, there are cases where the baseline is not learning at all whereas the Surgeon is able to overcome the initial local minimum.

We will subsequently highlight a few interesting cases.

Topology rescue. We recall that the small starting topology, consisting of only one hidden layer with 10 neurons, was purposefully chosen to be insufficient for convergence. In fact, as we can see in Fig. 2, the small topology baseline learns for neither SVHN, CIFAR-10, nor EuroSAT. The Surgeon on average manages to improve the topology and learn at least a little, even reaching a validation accuracy above 50% in case of the EuroSAT dataset. In case of the SVHN dataset, the global average over all runs contains several instances where even with the aid of the Surgeon, the model is not able to learn at all and stays stuck in the initial state, as well as a number of runs where the Surgeon is able to very quickly leave this local sink and then in fact provides a model that learns very well.

In Fig. 3 (left), we can see a single run of the Surgeon using the EuroSAT dataset and small starting topology, where an early *Add Layer* operation allows the network to train properly. The total parameter increase in this case is less than 1%, with the Surgeon preferring to add several small layers rather than widening existing layers. Note that due to the shape of our training data and choice of starting topology, a large portion of the network's parameters is required to connect the input layer to the first hidden layer. Adding a whole layer after the first 10-unit layer causes only a small overall increase, whereas widening the first

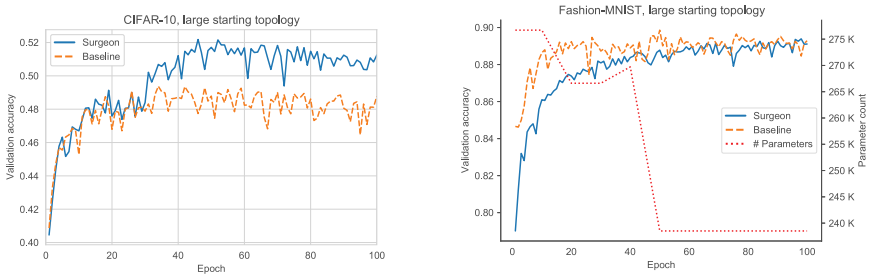


Fig. 4. Left: Single run of the CIFAR-10 dataset and large starting topology. The Surgeon manages to overcome the local optimum at around epoch 30. Right: Single run of the Surgeon on the Fashion-MNIST dataset and large starting topology. The Surgeon is able to reduce the total parameter count by around 14% while still reaching the same overall validation accuracy.

hidden layer might cause a greater increment in parameter count. Fig. 3 (right) shows the evolution of the topology over the course of the run.

Accuracy increase. The Surgeon is able to detect and improve sub-optimally sized network architectures. This works in cases such as above (cf. Fig. 3 left), where it is very obvious that little to no learning happens, but also in less apparent ones, where the base topology does learn to a certain extent. In Fig. 4 (left) we can see a single run of the Surgeon trained on CIFAR-10 where both the large starting topology as well as the Surgeon start learning in a similar fashion and soon reach a plateau. After a while however, the Surgeon is able to perform an *Add Layer* operation that allows the topology to overcome the local optimum which had been reached.

Parameter reduction. As mentioned previously, using the large topology as a starting point for the Surgeon does not improve achieved accuracy by any large margin (cf. Table 3). In Fig. 4 (right) we can see a single run of the Surgeon on the Fashion-MNIST dataset using the large starting topology. The baseline in this case is already performing quite well given that we are regarding a very basic network architecture. In this case we can observe pruning by the Surgeon, such that an early *Remove Layer* operation allows parameter reduction of almost 15%.

For the large starting topology, the composite scoring function given in Eq. (16) prevents any big jumps in accuracy since they would most likely come at the cost of a (potentially rather large) increase in network size. The scoring function much rather prefers network operations that keep the accuracy more or less constant while reducing network size.

Computational costs. In our trials the Surgeon is configured to produce a resulting topology that has been trained for exactly 100 epochs, with decision points every 10 epochs. We allow for a maximum of two concurrent branches, as well as re-drawing from potential branches up to two times, cf. Appendix. On average, 0.56 re-draws are necessary per decision step, resulting in an average total training amount of around 290 epochs per run of the Surgeon. The additional overhead produced by the analysis and modification modules mostly relies on matrix calculations and manipulations, which are highly optimized by standard python modules such as `numpy`, and thus cause only negligible additional computation time. In particular, making use of large GPU clusters is not a necessary requirement for running the Surgeon. For verification purposes and in order to ensure scalability of our code we ran our experiments both on a standard office computer without making use of any GPU acceleration, as well as a GPU cluster (cf. Appendix for detailed hardware specifications).

We can see from Fig. 2 that 100 epochs in most cases seem to be longer than is necessary for the Surgeon to reach convergence. With appropriate early stopping techniques, and/or a more dynamic training schedule, the total training amount for the Surgeon could be considerably reduced, and the resulting model (including its trained weights) used either as is, or further fine tuned (cf. Fig. 5).

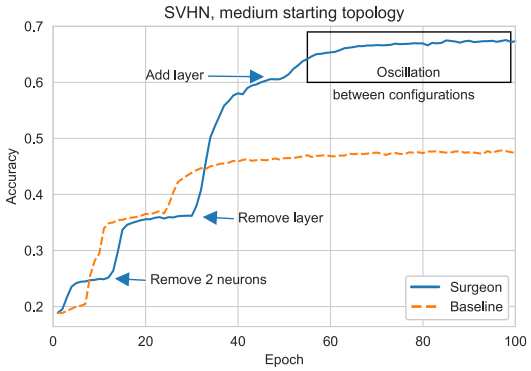


Fig. 5. Example of a topology evolution performed by the Surgeon on the SVHN dataset and medium starting topology. Early on, reducing the network size helps to increase the accuracy level (at epochs 10 and 30). Adding a whole layer in epoch 50 is still able to achieve some increase in accuracy. From epoch 60 on the network oscillates between very similar states. Ideally this behaviour can be used as an indicator for early stopping in future versions of the Surgeon.

5. Discussion and outlook

In this paper, we presented the Surgeon, a ANN/evolutionary algorithm hybrid optimization designed for neural architecture search. The algorithm utilizes a modification module, that is able to perform minimally invasive network surgeries, where the topology of the network is modified with as little change to overall input–output-behaviour as possible. Additionally, it uses a recommendation module, that analyses a given neural network and indicates which structural changes may be most beneficial. Those changes can be either increases of network width or depth in case of high information density, or respective decreases, in case network size can be reduced without fear of too high an accuracy loss. Both modules do not utilize any black box behaviour but are based on mathematical tools, which we presented in Section 3.

We put the Surgeon to the test on several combinations of starting topologies and datasets. We saw that the network generated by the Surgeon is able to outperform the baseline in case of suboptimal topologies, or reach comparable accuracies while pruning the underlying network structure to less resource-intensive topologies.

A very important feature of the Surgeon is that it itself is computationally cheap, with little overhead compared to simple baseline training. Via hyperparametre settings, it is possible to make use of larger computational power if required/available.

5.1. Limitations of this work and future goals

For this proof of concept work, we limited ourselves to the most basic and ubiquitous network structures — dense layers linked strictly in sequence. While such neural networks are best studied and easiest to understand and manipulate from a mathematical perspective, there are some draw backs.

Fully connected neural networks require a fixed input shape, thus they are not suited for a variety of classical deep learning tasks such as natural language processing (NLP) or image detection. Large tabular datasets often contain ordinal or categorical variables which are not well suited for deep learning tasks as gradient calculation becomes somewhat ill-defined. In particular, we are mostly limited to making use of image classification benchmark sets. For these, state of the art is driven by large, computationally expensive algorithms that allow more complex topological elements such as convolutional or recurrent layers, skip connections, etc., see for example Liu et al. (2019), Xu et al. (2020) and Zoph and Le (2017). In general, few benchmark results exist for fully connected neural networks trained on these datasets.

For future work we wish to extend and improve both the Surgeon as well as the underlying modules. Currently, the Surgeon follows a static routine, with hyperparameters such as epochs steps, number of selected candidates, or number of concurrent branches staying constant throughout the whole process. This could be changed to a more dynamic approach, and could maybe integrate further features such as adaptive learning rates, or a more sophisticated memory, as well as early stopping mechanisms to improve performance gains compared to baseline training even further. The recommendation module can be improved by finding a closer approximation for the BIC. For larger starting topologies, the scoring function given in Eq. (16) seems to be chosen too restrictively. The balancing between the different components can be adapted to allow for more drastic additions even when the network is already quite large to begin with.

Lastly we want to expand the modification module to allow more complex topologies as well, and include an option to cross architecture types. This would allow us to include e.g. convolutional or recurrent elements, change a network from one type to another, or even freely mix and match as required.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix. Hyperparameter settings and machine specifications

Simulations for the SVHN and CIFAR-10 datasets were performed on a Windows 10 machine with an Intel(R) Core(TM) i7-9700K CPU 3.6 GHz processor and 64,0 GB RAM. The code was implemented in python 3.7.7 using tensorflow 2.1.0.

Simulations for the CIFAR-100, EuroSAT, EMNIST, and Fashion-MNIST datasets were performed on a virtual machine running on a 24-core 2.1 GHz Intel Xeon Scalable Platinum 8160 processor, which is equipped with a Tesla V100 GPU card with 16 GB memory. The code for these datasets was ported to python 3.8.2 using tensorflow 2.3.0 and tensorflow-datasets 4.1.0.

We chose the following hyperparameters:

The modification module.

- Amount of re-training per modification type:
  - Identity: 1 · re-training batches
  - Add Neuron: 1 · re-training batches
  - Add Layer: 10 · re-training batches
  - Remove Neuron: 10 · re-training batches
  - Remove Layer: 10 · re-training batches
  - Truncated SVD: 1 · re-training batches

The recommendation module.

- Absolute singular value threshold: $\epsilon_2 = 0.3$
- Relative singular value threshold: $\epsilon_1 = 0.005$
- Recommendation style: best scoring per modification type
- Include additional random draw: True

The Surgeon.

- Training time for winning branch: 100 Epochs
- Initial pre-training: 10 Epochs
- Interval between decision points: 10 Epochs
- Concurrent branches: 2
- Maximum re-draws per decision step: 2
- Number of batches for re-training: 25 batches

The testing setup.

- Random seeds: 6, 63, 72, 77, 97
- Loss function: Sparse categorical crossentropy
- Optimizer: SGD
  - learning rate: 0.005
- Metrics: accuracy
- Batch size: 16

References

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? In I. Dhillon and D. Papailiopoulos and V. Sze (Eds.), *Proceedings of machine learning and systems, Vol. 2* (pp. 129–146).

Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018). Efficient architecture search by network transformation. In *32nd AAAI conference on artificial intelligence, AAAI 2018* (pp. 2787–2794). AAAI Press, arXiv:1707.04873.

Chen, T., Goodfellow, I. J., & Shlens, J. (2016). Net2Net: Accelerating learning via knowledge transfer. In Y. Bengio, & Y. LeCun (Eds.), *4th international conference on learning representations, (ICLR)*.

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. arXiv:1702.05373.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 1269–1277). Curran Associates, Inc..

DiMattina, C., & Zhang, K. (2010). How to modify a neural network gradually without changing its input-output functionality. *Neural Computation*, *22*(1), 1–47. http://dx.doi.org/10.1162/neco.2009.05-08-781.

Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1–21.

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th international conference on learning representations, (ICLR)*.

Girshick, R. (2015). Fast r-CNN. In *2015 IEEE international conference on computer vision (ICCV)* (pp. 1440–1448). IEEE Computer Society, http://dx.doi.org/10.1109/ICCV.2015.169.

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2018). Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. In *IGARSS 2018-2018 IEEE international geoscience and remote sensing symposium* (pp. 204–207). IEEE.

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. http://dx.doi.org/10.1109/CVPR.2017.243.

İrsoy, O., & Alpaydın, E. (2020). Continuously constructive deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(4), 1124–1133. http://dx.doi.org/10.1109/TNNLS.2019.2918225.

Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., & Xing, E. P. (2018). Neural architecture search with Bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 2016–2025). Curran Associates, Inc..

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*: *Tech. rep*., University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeNail, A. (2019). NN-SVG: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, *4*(33), 747. http://dx.doi.org/10.21105/joss.00747.

Li, L., Khodak, M., Balcan, N., & Talwalkar, A. (2021). Geometry-aware gradient algorithms for neural architecture search. In *9th international conference on learning representations (ICLR)*.

Li, L., & Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. In *Conference on uncertainty in artificial intelligence (UAI)*.

Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *7th international conference on learning representations (ICLR)*.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on genetic algorithms* (pp. 379–384). Morgan Kaufmann Publishers Inc..

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. http://dx.doi.org/10.1214/aos/1176344136.

Wang, R., Cheng, M., Chen, X., Tang, X., & Hsieh, C.-J. (2021). Rethinking architecture selection in differentiable NAS. In *9th international conference on learning representations (ICLR)*.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv:cs.LG/1708.07747.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., et al. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *8th international conference on learning representations (ICLR)*.

Xue, J., Li, J., & Gong, Y. (2013). Restructuring of deep neural network acoustic models with singular value decomposition. In *Proceedings of the annual conference of the international speech communication association (INTERSPEECH)* (pp. 2365–2369).

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *5th international conference on learning representations (ICLR)*.

A.2 Publication 2

Predicting the inhibition efficiencies of magnesium dissolution modulators using sparse machine learning models

E. J. Schiessler, T. Würger, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler and R. C. Aydin

E.J.S., T.W., S.V.L., R.H.M., C.J.C., M.L.Z., C.F., and R.C.A. contributed to the conception and design of the study. C.F. generated the molecular descriptor database. E.J.S. did the theoretical analyses and wrote the supporting code. E.J.S., T.W., R.C.A., and C.F. evaluated the quality of the presented models. E.J.S. and T.W. created the figures. E.J.S. and C.F. wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

Reprinted from E. J. Schiessler, T. Würger, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler, and R. C. Aydin. Predicting the inhibition efficiencies of magnesium dissolution modulators using sparse machine learning models. *npj Computational Materials*, 7(1):193, December 2021. 10.1038/s41524-021-00658-7, *licensed under the Creative Commons Attribution 4.0 License* (http://creativecommons.org/licenses/by/4.0/).

npj | Computational Materials

www.nature.com/npjcompumats

ARTICLE OPEN

Check for updates

Predicting the inhibition efficiencies of magnesium dissolution modulators using sparse machine learning models

Elisabeth J. Schiessler[1], Tim Würger[2,3], Sviatlana V. Lamaka[2], Robert H. Meißner[2,3], Christian J. Cyron[1,4], Mikhail L. Zheludkevich[2,5], Christian Feiler[2]✉ and Roland C. Aydin[1]✉

The degradation behaviour of magnesium and its alloys can be tuned by small organic molecules. However, an automatic identification of effective organic additives within the vast chemical space of potential compounds needs sophisticated tools. Herein, we propose two systematic approaches of sparse feature selection for identifying molecular descriptors that are most relevant for the corrosion inhibition efficiency of chemical compounds. One is based on the classical statistical tool of analysis of variance, the other one based on random forests. We demonstrate how both can—when combined with deep neural networks—help to predict the corrosion inhibition efficiencies of chemical compounds for the magnesium alloy ZE41. In particular, we demonstrate that this framework outperforms predictions relying on a random selection of molecular descriptors. Finally, we point out how autoencoders could be used in the future to enable even more accurate automated predictions of corrosion inhibition efficiencies.

npj Computational Materials (2021)7:193 ; https://doi.org/10.1038/s41524-021-00658-7

INTRODUCTION

Magnesium (Mg) is among the most abundant elements on our planet[1] and exhibits a high potential to revolutionize light metal engineering in a large number of application fields. Key to unlocking the full potential of Mg is to control the surface reactivity characteristics of the material due to the relatively high electrochemical reactivity of Mg, where each application field imposes unique challenges. Corrosion needs to be prevented in transport applications[2–4] (e.g., aeronautics and automotive) to ensure the integrity of the material. Medical applications (e.g., temporary, biodegradable bone implants)[5,6] require a degradation rate tailored to a patient-specific injury to support recovery. Batteries with a Mg anode[7,8] need a steady dissolution rate to keep the output voltage constant. Fortunately, small organic molecules exhibit great potential to control corrosion in these highly versatile application areas—due to their almost unlimited chemical space. Each service environment fundamentally changes the boundary conditions to achieve the above mentioned goals, as the small organic molecules are usually incorporated in a complex coating system for transport applications, whereas they become a solute component of the electrolyte for Mg-air batteries.

Despite impressive progress in the screening of potential additives by efficient high throughput techniques[9–12], experimental approaches alone cannot possibly explore more than a tiny fraction of the vast space of compounds with potentially useful properties. However, data-driven computational methods[13–21] can explore large areas of chemical space orders of magnitude faster, and can thus be exploited to preselect promising chemicals prior to deep experimental testing. Concomitantly, computational techniques[22–27] can be utilized to unravel the underlying chemical mechanisms of corrosion and its inhibition, which in turn provide additional input features for predictive quantitative structure-property relationship (QSPR) models.

Naturally, data-driven methods cannot make reliable predictions for molecules outside the domain of their respective training data (e.g., for compounds that exhibit functional groups or elemental species not present in the training set). Hence, the dataset employed for training has to reflect the complexity of the relevant chemical environment, and should ideally be a large, reliable, as well as chemically diverse and balanced database to enable accurate and robust predictions for a broad range of materials. However, the versatility of the vast chemical space of interest is associated with a wide range of different functional moieties and molecular features, and renders it challenging to identify meaningful input features to develop predictive models with a wide applicability. Cheminformatics software packages like alvaDesc[28] and RDKit[29] provide a large variety of molecular descriptors ranging from structural and topological features to more complex input features like molecular signatures[30] and molecular fingerprints. Furthermore, advances in computing power and simulation algorithms over the last decades enabled multiscale simulations (density functional theory calculations, molecular dynamics simulations, and finite element modeling)[25,26,31–37], thus providing even more potentially useful molecular descriptors for the training of data-driven models[18,38]. Additional sets of molecular descriptors might be based on properties of the used material as well as information on the service environment.

The quality of predictive models substantially depends on the selected molecular descriptors, as input features with low relevance to the target property will degrade the model. Especially the correlation (or its absence) of descriptors derived from computer simulation with the experimental performance of corrosion inhibiting agents is controversially discussed[15,39–41].

[1]Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Geesthacht, Germany. [2]Institute of Surface Science, Helmholtz-Zentrum Hereon, Geesthacht, Germany. [3]Institute of Polymers and Composites, Hamburg University of Technology, Hamburg, Germany. [4]Institute for Continuum and Material Mechanics, Hamburg University of Technology, Hamburg, Germany. [5]Institute for Materials Science, Faculty of Engineering, Kiel University, Kiel, Germany. ✉email: christian.feiler@hereon.de; roland.aydin@hereon.de

Published in partnership with the Shanghai Institute of Ceramics of the Chinese Academy of Sciences

npj nature partner journals

However, it was demonstrated that they can be highly relevant in models that combine them with input features derived from the molecular structure[18]. Statistical methods such as analysis of variance (ANOVA) are well established, computationally cheap tools for the identification of relevant features and parameters[42–45], but may struggle to capture intricate dependencies between variables. This problem can be overcome by machine learning techniques for sparse feature selection[14,46–49].

In this paper, we propose and compare two different sparse feature selection strategies: statistical analysis using ANOVA *f*-tests[42–45] and recursive feature elimination based on random forests[47,49–52], using training data for the Mg alloy ZE41. The training data relate results of density functional theory (DFT) calculations and molecular descriptors generated by the alvaDesc cheminformatics software package to known corrosion inhibition efficiencies of chemical compounds. We demonstrate how our feature selection strategies can be combined with deep learning into a sparse, predictive QSPR (quantitative structure-property relationship) framework. Moreover, we demonstrate how in this context autoencoders[53,54] can be used for contour maps and anomaly detection.

RESULTS AND DISCUSSION

The software package alvaDesc was utilized to generate a set of 5290 potential input features for our model. The obtained values were divided into different subcategories, ranging from counts of simple structural features of molecules to arcane descriptors derived from chemical graph theory. After removing all molecular descriptors that exhibited constant values or were essentially zero, 1254 descriptors remained and were augmented by six molecular descriptors derived from DFT calculations (*cf.* "Methods" section). In the resulting set of 1260 molecular descriptors (features), we searched for those input features with the greatest impact on the corrosion inhibition responses of 60 small organic molecules on ZE41 (target). A list of the considered molecules can be found in Supplementary Table 7, along with their SMILES strings and experimentally determined inhibition efficiencies. We only used data of dissolution modulators from our experimental database[55] with a molecular weight of less than 250 Da that were employed at a concentration of 0.05 M.

For sparse feature selection, we applied two different approaches: The first one was based on individual feature selection via an *f*-test based analysis of variance (ANOVA) to analyse the individual importance of the different molecular descriptors. The second one was a grouped feature selection approach utilizing recursive feature elimination with random forests as the underlying regressor to analyse the importance of *n*-tuples of molecular descriptors. For a detailed description of the applied methods, *cf.* "Methods" section. We chose to look for the top 3, 5, and 63 (equivalent to 5%) most relevant features respectively, and repeated each approach multiple times to overcome any bias induced by specific random seeds. To evaluate and compare the selection methods, as well as the predictive power of the selected features, we trained several deep learning models using the identified molecular descriptors as respective sole inputs. As an additional baseline for comparison we used models trained on randomly selected features, as well as a model trained on the full dataset.

All analyses and trainings were performed with a reduced dataset, where a randomly chosen 10% of samples (i.e., six samples, *cf.* Table 3) were withheld and subsequently served as a representative example of completely unknown validation data. Furthermore, a full 10-fold cross validation was performed on all deep learning models. An overview of the workflow is depicted in Fig. 1.

Individual feature selection

First, we utilized an *f*-test based ANOVA algorithm to rank each molecular descriptor according to its individual significance for predicting the inhibition efficiencies of ZE41 via its *f*-score. Features may be deemed significant if their score is ≫1.

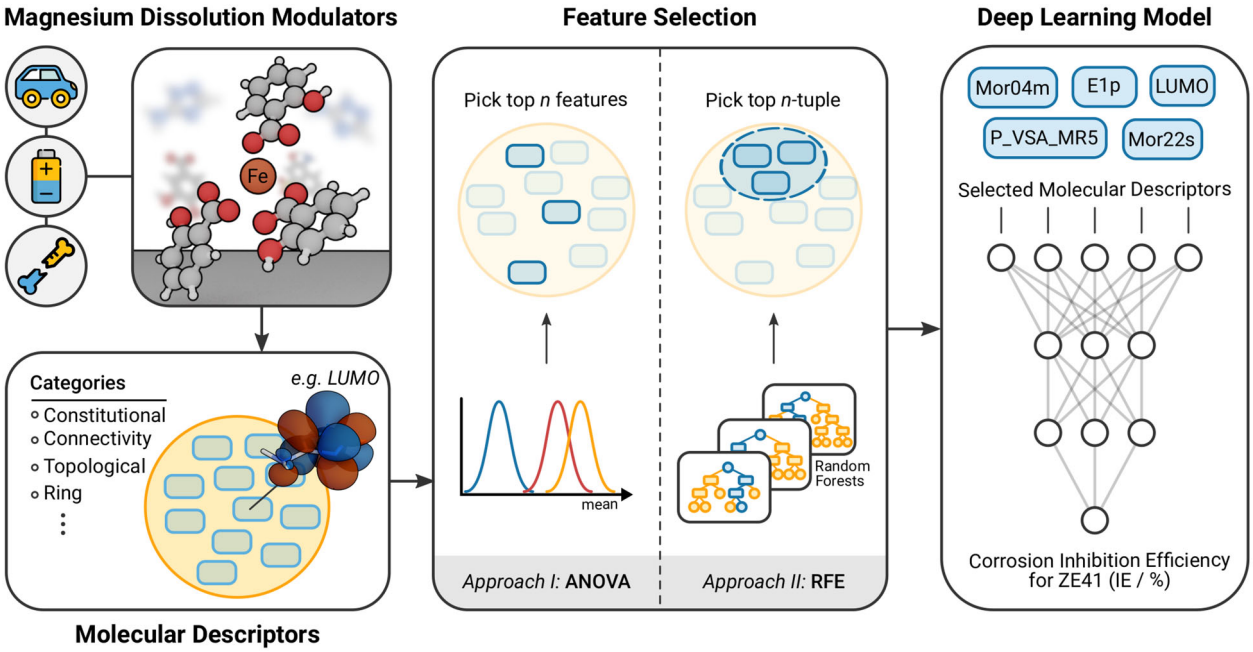


Fig. 1 Workflow overview. Our overarching objective is the prediction of the magnesium corrosion inhibition efficiency of different molecular dissolution modulators. To this end, first relevant molecular features are selected either by (approach I) analysis of variance (ANOVA) or by (approach II) recursive feature elimination based on random forests (RFE), a type of machine learning. The best-performing feature set defines the input for a deep learning model. This model allows the desired predictions of quantitative structure-property relationships (QSPR) for the efficiency of magnesium dissolution modulators, in our study specifically for the magnesium alloy ZE41.

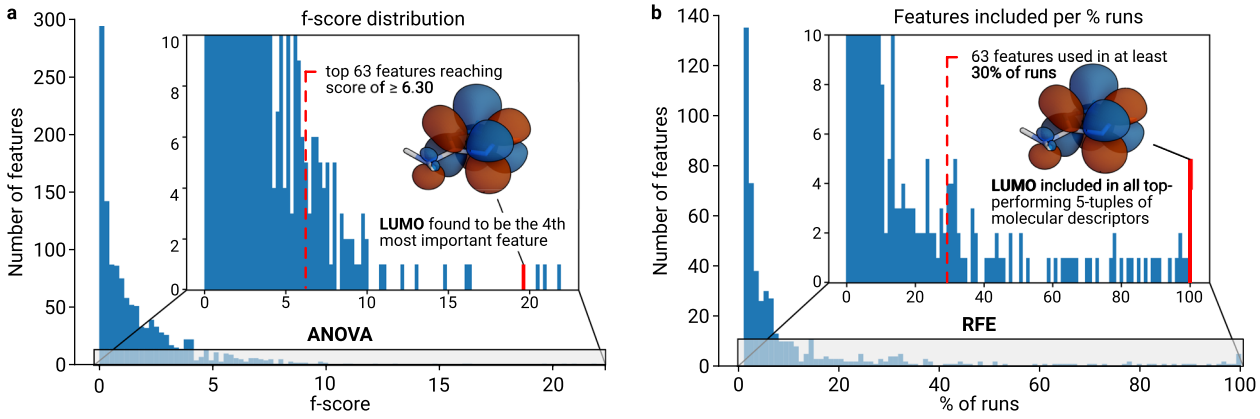


Fig. 2 Feature distributions. a Distribution of *f*-scores as calculated by ANOVA. The top 63 features reach a score of 6.3 or higher, with only 11 features scoring 10 or above. **b** The recursive feature elimination (RFE) identifies a total of 504 features over a series of 100 runs with random initialization as potential candidates for a top 63-tuple. Selecting among them those identified in at least 30% of the runs (frequency analysis) can be used to define the most relevant 63 features.

One can define the *n* top scoring molecular descriptors by simply ranking them via their *f*-score.

In our study we observed *f*-scores in the range from 0 to 21.92, with the vast majority of features (≈92%) scoring below 5, *cf*. Fig. 2a. Selecting the top 3, 5, or 63 (i.e., 5%) features translates to *f*-score thresholds of 19.7, 16.1, and 6.3, respectively (corresponding to *p*-value thresholds of 0.00004, 0.00019, and 0.0155 respectively). The top five identified descriptors are CATS2D_03_AP, CATS3D_03_AP, CATS3D_02_AP, LUMO/eV, and P_VSA_MR_5 (in descending order of relevance), for the set of top 63 descriptors *cf*. Supplementary Table 2.

In particular, one of the included DFT-derived input features, the lowest unoccupied molecular orbital energy levels (LUMO), was identified as one the most relevant descriptors. Three of the five most relevant input features belong to the class of CATS descriptors[56,57], which are linked to properties of potential pharmacophores and are related to the discovery of novel drugs, since they indicate whether a ligand is likely to bind to a receptor site of a biological macromolecule[58]. They also seem to encode structural information on functional moieties that are capable of forming coordinative bonds with ions of interest, rendering them highly relevant for the development of our model, as the inhibition efficiency of the small organic molecules is strongly dependent on their capability to form complexes with Mg^{2+} and $Fe^{2+/3+}$. The P_VSA class is comprised of 2D descriptors that reflect the sum of atomic contributions to the van-der-Waals surface area[59]. The P_VSA descriptor identified by the ANOVA approach is related to the polarizability of the chemicals in our data set.

Grouped feature selection

As the interplay and correlations between parameters can have a significant impact on the quality of the prediction, it may not be sufficient to merely select the individually most predictive features and use them as the combined input for a predictive model[47]. Therefore, we additionally identified the 3-tuples, 5-tuples and 63-tuples of *grouped* most relevant features via recursive feature elimination (RFE) using random forests. We performed 100 runs of RFE with varying random seeds, where a random forest consisting of 100 trees was trained in each run. Subsequently, the *n*-tuples that won most often were selected to be the most relevant grouped features with LUMO, P_VSA_MR_5, Mor04m (selected in 83/100 runs) for the 3-tuples and LUMO, P_VSA_MR_5, Mor04m, E1p, Mor22s (selected in 21/100 runs) for the 5-tuples. It is noteworthy, that the energy level of the lowest unoccupied molecular orbital (LUMO) of the compounds in the training set, which was derived from DFT calculations, was again among the most relevant features, along with P_VSA_MR_5. Furthermore, different descriptors belonging to the class of 3D-MoRSE (Molecular Representation of Structures based on Electron diffraction) were selected[60,61]. These are abbreviated as "Mor" and are a mathematical representation of XRD patterns where the obtained signals can be weighted by previously discussed schemes. E1p belongs to the class of WHIM descriptors which are 3-dimensional descriptors that collect information about size, shape, symmetry, and atom distribution of the molecule. E1p is related to the atoms distribution and density around the origin and along the first principal component axis. The index *p* indicates that the selected descriptor is calculated by weighting the atoms with their polarizability value.

In case of the 63-tuple, no group was found to be inherently most relevant. We therefore artificially constructed the most relevant group by a frequency analysis of all features that were included at least once in any of the RFE runs. Among these 504 features, interestingly only the ones in the top 5-tuple occurred in every single run. 135 features (≈27%) were identified just once, and 302 (≈60%) were found to be in at most 5 supports. The top 63 features were included in at least 30% of all runs, *cf*. Fig. 2b, for the full list *cf*. Supplementary Table 3.

This underlines that molecular descriptors derived from quantum mechanical calculations can be highly relevant input features for models that predict the corrosion inhibition efficiency of small organic molecules for Mg alloys. This is in good agreement with our findings for commercially pure Mg containing 220 ppm iron impurities (CPMg220), where the frontier orbital energy gaps exhibited moderate correlation with the corresponding inhibition efficiencies[18], and could be utilized to obtain a robust predictive model in combination with structural input features. The results of others[39–41] suggest that his type of descriptor should be taken with care because its relevance may be compromised if not combined with structural input features. Yet, as demonstrated also by our above results, if properly used, it can be a highly powerful feature for the prediction of corrosion inhibition efficiency.

To counter potential bias from the choice of the validation set when selecting the most relevant input features, we performed a 5-fold cross validation, i.e., selecting a different validation set and repeating the whole feature selection process using RFE described above independently five times. Due to computational limitations, we performed this cross-validation only on the grouped 5-tuple of features, as results obtained from subsequent deep learning models suggest an optimal trade-off between a low number of input features and a low computational cost on the one hand and

Table 1. Median statistics over the full 10-fold cross validation.

No. of features	3			5			63			1260
Model type	Tiny model			Small model			Medium model			Large model
Selection method	*a*	*b*	*c*	*a*	*b*	*c*	*a*	b	c	
RMSE/pp	66	56	66	57	50	66	50	50	56	63
R^2	0.40	0.51	0.30	0.68	0.66	0.41	0.60	0.62	0.50	0.35
Pearson's *r*	0.61	0.71	0.54	0.82	0.81	0.64	0.77	0.79	0.70	0.59
p-value	0.23	0.13	0.27	0.05	0.06	0.17	0.08	0.06	0.12	0.22

Median values of root mean squared errors (RMSE), coefficients of determination (R^2), correlation coefficients (Pearson's *r*) and *p*-values of the full 10-fold cross validation for all trained models by model type and feature selection method (a: ANOVA, b: RFE, c: random selection).

a high predictive accuracy on the other hand. The initially identified top performing 5-tuple of molecular descriptors was confirmed by this cross-validation, along with two other 5-tuples, all three of which agreeing on four out of five descriptors. The first of the three identified sets consists of LUMO, P_VSA_MR_5, Mor04m, E1p, HOMO, the second one of LUMO, P_VSA_MR_5, Mor04m, E1p, Mor22s and the third one of LUMO, P_VSA_MR_5, Mor04m, E1p, CATS3D_02_AP.

Predictive models using deep learning techniques

From the ranked list of individually most relevant features (selected by ANOVA), we used the top 3, 5, and 63 molecular descriptors to train three deep learning models, from here on called M3a (tiny model), M5a (small model), M63a (medium model). We performed a complete 10-fold cross validation, i.e., we split the dataset into ten equal parts (folds) and subsequently withheld one fold as a test set, while the rest of the data served as training set. On each fold, every model was trained 100 times with varying random seeds to obtain results largely independent on specific random initializations. Subsequently, we repeated the same procedure with the top 3, 5, and 63 most relevant molecular descriptors obtained by grouped feature selection via RFE as input for the three neural network models M3b (tiny model), M5b (small model), and M63b (medium model).

Finally, we selected 3, 5, and 63 random molecular descriptors to train three neural network models M3c (tiny model), M5c (small model), M63c (medium model) as a reference base line to assess the quality of the aforementioned models M3a, M3b, M5a, M5b, M63a, and M63b. The input features for these models were re-drawn from the set of 1260 available features in each of the 100 training runs.

As an additional baseline we trained a deep neural network M1260 (large model) which uses all available molecular descriptors as its input. This model can be considered the joint limit case of the above three feature selection methods ANOVA, RFE, and random in case that the number of selected features is increased to its maximal value of 1260.

In Table 1 we report for all the above neural network models median values (across the ten folds) of four key statistical measures of their predictive capabilities, that is, of the root mean squared error RMSE (given in percentage points), the coefficient of determination R^2, Pearson's correlation coefficient *r*, and the *p*-value. In Table 1 we observe several consistent trends. First, all statistical measures of predictive capability noticeably improve when the number of input features is increased from 3 to 5 to 63 for all the three feature selection methods (ANOVA, RFE, or random). Second, the two sparse feature selection procedures (ANOVA and RFE) consistently outperform in all measures a simple random feature selection, which underlines their practical value. Third, the two sparse feature selection procedures (ANOVA and RFE) exhibit a similar performance, with RFE slightly outperforming ANOVA with respect to RMSE, which can in many respects be considered the most relevant one of the four statistical measures. This underlines that grouped features selection has indeed—as one would also expect—advantages over individual features selection, though at least in the framework used herein only to a limited extent. A fourth important observation is the decline of performance when increasing the number of input features to 1260. This can be understood from the fact that such unspecific input dilutes the relevant information harbored by the input in a way that makes systematic learning of QSPR more difficult. Quite interestingly, for the two sparse features selection methods (ANOVA and RFE)—unlike for the random feature selection—the performance already stagnates when increasing the number of input features from 5 to 63, indicating that they can help to identify a very small group of features that carries nearly the whole information relevant for predictions.

It is noteworthy that even when using a sparse feature selection method, the error of the predictions based on the selected features still remains substantial. While fully overcoming this problem would go beyond the scope of this paper, we further investigated into the reasons of this problem. Analysing our data we found that the performance of predictions based on sparse feature selection is substantially adversely affected by only a few outliers. To illustrate this, we consider more closely compound no. 13, 3,5-Dinitrobenzoic acid. Unlike all the other 59 molecules in our data base, it contains an NO_2 functional group. This important chemical difference is supposedly the reason why the information carried by the other compounds cannot help a neural network to make accurate predictions also for 3,5-Dinitrobenzoic acid, which indeed results in a very large error for any of the above introduced predictive neural networks. Naturally, such a large error affects the otherwise very good performance of predictions based on sparse feature selection methods much more adversely than the generally much less accurate predictions based on randomly selected features. To demonstrate this, we show in Table 2 the results for one specific fold where we manually removed from the validation set 3,5-Dinitrobenzoic acid. Evidently, this substantially improves the predictions in particular made on the basis of grouped feature selection, while the quality of predictions based on of tiny or small sets of randomly selected features remains rather limited. Detailed information about the fold, validation and neural network predictions underlying to Table 2 are presented in Supplementary Tables 5 and 6. We performed Pearson correlation tests for all different models presented in Table 2 and observed in particular for neural networks receiving input features obtained from grouped feature selection a positive correlation coefficient of 0.97 and significant *p*-values below 0.01. Figure 3 illustrates the performance of the deep neural networks M5b and M1260 for the (reduced) validation set discussed in Table 2.

Table 2. Statistics on the representative validation set.

No. of features	3			5			63			1260
Model Type	Tiny model			Small model			Medium model			Large model
Selection method	*a*	*b*	*c*	*a*	*b*	*c*	*a*	*b*	*c*	
RMSE/pp	50	24	66	51	26	60	49	23	40	38
R^2	0.56	0.94	0.53	0.54	0.94	0.95	0.58	0.94	0.94	0.95
Pearson's *r*	0.75	0.97	0.73	0.73	0.97	0.97	0.76	0.97	0.76	0.97
p-value	0.14	0.01	0.16	0.16	0.01	0.01	0.14	0.01	0.08	0.01

Root mean squared errors (RMSE), coefficients of determination (R^2), correlation coefficients (Pearson's *r*) and *p*-values of the representative validation set predictions for all trained models by model type and feature selection method (a: ANOVA, b: RFE, c: random selection). 3,5- Dinitrobenzoic acid (compound No. 13) was omitted for calculation of the statistical values as its molecular features are in parts substantially outside of the domain covered by the training data.

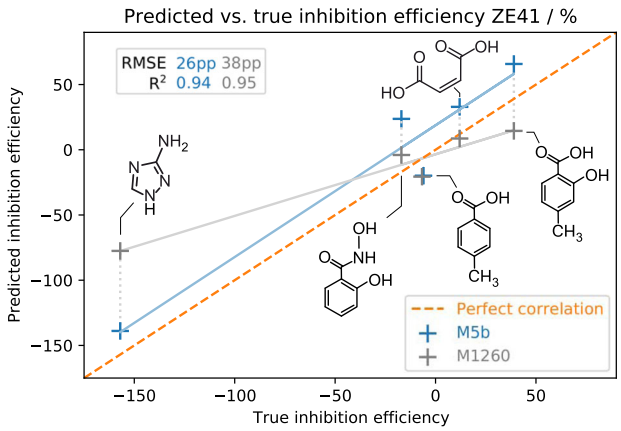


Fig. 3 Model performance. Predicted vs. true target values for the validation sets as obtained by M5b and M1260. Linear regressions are depicted as accordingly colored lines. 3,5-Dinitrobenzoic acid was excluded as it contains features that are outside of the domain of the trained model.

Comparing the median performance over all cross validation folds with that on the representative validation set showcases the potential of predictive modeling when combined with appropriate outlier detection methods. As pointed out above, a few outliers can have a drastic impact on the quality of the predictive models. In particular, one of the ten cross validation folds contains outliers that consistently yielded very poor results across all models and metrics. For this reason we elected to present median rather than mean values across all statistics, for the corresponding mean values table *cf.* Supplementary Table 4. Besides outlier detection, repeating the feature selection process for each model and each fold can also increase performance.

Autoencoders

So-called autoencoders are a type of neural network that is not used for predictions but rather to learn a lower-dimensional representation (code) of the input data, from which the original input can be reconstructed as accurately as possible (*cf.* "Methods" section). Herein we applied an autoencoder with a code of dimension 2 to the 5-tuple of features determined by grouped feature selection. The resulting two-dimensional representation of the 60 chemical compounds studied herein is plotted in Fig. 4a. Subsequently, we used the decoder part of the autoencoder to generate a contour map of predicted inhibition efficiencies across the whole two-dimensional reduced feature space, Fig. 4b to make anomalies even easier to spot with the naked eye. It is immediately noticeable as a prominent anomaly in the plot of the reduced (two-dimensional) feature space that there are two samples with a highly negative inhibition efficiency within a cluster of samples with a (moderately) positive inhibition efficiency. The first one is 4-hydroxybenzoic acid with an inhibition efficiency of −170% whose parent system salicylic acid causes a considerably higher inhibition efficiency of 37% despite very similar molecular features. Addition of another hydroxyl group in 3,4-dihydroxybenzoic acid (the second outlier) leads to a further increase of the Mg^{2+} binding ability resulting in an inhibition efficiency of −270%. The behavior of the latter can be attributed to the significantly higher stability constant of the corresponding complex of 3,4-dihydroxybenzoic acid with Mg ($\log K(Mg^{2+}) = 9.84$) in comparison to that of salicylic acid ($\log K(Mg^{2+}) = 4.7$)[62,63]. We assume that a similar effect is the reason for the unique behavior of 4-hydroxybenzoic acid although there is no stability constant available in the literature to support this claim. Additionally, the corresponding ligands do not only shift dissolution equilibria, but they also compete with OH^- for binding Mg^{2+} thus preventing the formation of a semi-protective $Mg(OH)_2$ layer on the substrate. Consequently, 4-hydroxybenzoic acid and 3,4-dihydroxybenzoic acid are currently investigated concerning their potential as effective additives for Mg-air battery electrolytes.

In summary, we have pointed out above how sparse feature selection methods can help to identify those molecular descriptors that carry the most valuable information for predictions of the corrosion inhibition efficiency of organic molecules on the degradation of magnesium alloys. Our results clearly demonstrate that in addition to classical structural descriptors also those directly derived from DFT calculations can be highly relevant for data-driven predictions. Interestingly, our methods of spare feature selection reveal that the Chemically Advanced Template Search (CATS) descriptors form a particularly valuable basis for predictions. These are generally known to bear great potential for e.g., the AI-driven discovery of drugs[64]. Our results suggest that the pharmacophore properties encoded therein can also help to describe the capacity of small organic molecules to form complexes with metal ions like Mg^{2+} and $Fe^{2+/3+}$. This appears natural since atoms that may act as hydrogen bond acceptors (e.g., a nitrogen atom with a lone pair) may also act as donor for the formation of a coordinative bond in another context. In some cases an intuitive understanding of the relevance of descriptors selected above may be difficult. Yet, it is striking that the DFT-derived descriptor LUMO seems to play a significant role. This claim is corroborated by the outcome of both the individual and grouped feature selection. Our above analyses were not biased in any way by any expectation of specific features becoming dominant. Yet, LUMO was selected approximately 240 times more often by our smart feature selection algorithms than expected from random probability (*cf.* Supplementary Notes) which is a strong hint at a possible causal relationship between LUMO and

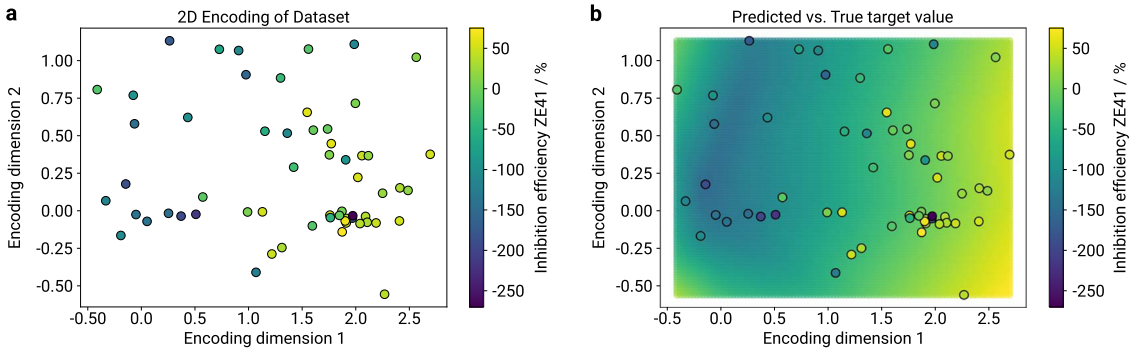


Fig. 4 Using autoencoders for outlier detection and contour maps. a Input features reduced to a two-dimensional code. **b** The decoder part in combination with an appropriate predictive model (such as a deep neural network) can be used to generate contour maps across the space spanned by the dimensions of the two-dimensional code.

the corrosion inhibition efficiency. Using the example of a specific fold within our 10-fold cross validation, we pointed out that the elimination or proper treatment of outliers can be expected to play a key role in further improving the accuracy of feature-based predictions of the corrosion inhibition efficiency. We showcased the ability of autoencoders to detect potential anomalies within datasets, which can be especially useful when working with small datasets. Note that the affected samples were included in all analyses and training steps as is. Yet, as apparent from the discussion above, it is very likely that the development of methods for a special treatment or at least detection of outliers could be an important step to improve data-driven predictions of corrosion inhibition efficiencies substantially, which opens up a promising avenue of future research.

METHODS

Molecular descriptor generation

To define molecular descriptors, we first determined the structures of the 60 chemical compounds of interest using the quantum chemical software package Turbomole 7.4.[65] at the TPSSh/def2SVP[66,67] level of density functional theory. Six of the molecular descriptors considered herein are directly derived from the output of the performed DFT calculations. These are the frontier orbital energies (HOMO, LUMO) as well as the frontier orbital energy gap (ΔE_{HL}), the calculated heat capacities (C_p, C_v) and the chemical potential (μ) calculated at 293 K. The thermodynamic properties were derived from the calculated vibrational frequencies using the Turbomole module *freeh* with default parameters for the calculations. The Cartesian coordinates resulting from our DFT calculations are subsequently used as input for the cheminformatics software package alvaDesc 1.0[28] to generate roughly 5000 molecular descriptors related to structural features. After omitting molecular descriptors with constant values and/or those that are close to zero, we used the remaining 1254 descriptors in combination with the above mentioned six DFT descriptors as input features for our sparse feature selection method.

Dataset preprocessing

We randomly selected 10% of the available data (i.e., six samples) using `scikit-learn`'s train-test-split[68] that are withheld from all further preprocessing, analysis, and training. These samples serve as an unknown validation set, and are used to validate the predictive abilities of the trained models. A representative validation set is illustrated in *cf.* Table 3. The index is used for numbering of the 60 chemical compounds of interest. We applied linear min-max scaling to all descriptors to map their values on the interval [−1, 1]. The target variable (corrosion inhibition efficiency) was mapped on the interval [0, 1].

Data analysis—individual features

To identify the most relevant molecular descriptors for predicting inhibition efficiency, we considered two approaches. The first was to regard each feature individually, and determine its influence on predicting the target variable, *i.e.* to look for the individually most relevant features. We did so by means of *f*-test based analysis of variance (ANOVA)[42–45]. An *f*-test (or *F*-test) is a test to see whether two independent, identically distributed variables X_1 and X_2 have the same variance. The *f*-score is given by $f = \sigma^2_{X_1}/\sigma^2_{X_2}$, with $\sigma^2_{X_i}$ denoting the variance of X_i. The null hypothesis may then be rejected if *f* is either below or above a chosen threshold *α*.

F-test based ANOVA calculates an f-score for every molecular descriptor compared to the target variable (corrosion inhibition efficiency). This score provides a statistic (with an F(1, *k*−2)-distribution, where *k* is the number of samples) for each descriptor for testing the hypothesis whether its distribution is the same one as the one of the target variable. The higher the *f*-score, the higher the presumed relevance of a descriptor. Herein, we used the top 3, 5, and 63 (i.e., 5%) descriptors as input for a subsequent deep learning framework.

Table 3. Representative validation set.

Index	Compound description	Inhibition efficiency ZE41/%
0	3-Amino-1,2,4-triazole	−157
5	4-Methylsalicylic acid	39
13	3,5-Dinitrosalicylic acid	38
36	Maleic acid	12
45	*p*-Toluic acid	−6
54	Salicylhydrocamic acid	−17

Set aside representative validation set (randomly selected).

Data analysis—grouped features

Those descriptors that individually hold the most amount of information need not necessarily work best together as a group when used as the input for a deep learning model. Thus we also identified *n*-tuples of features that are most relevant as a group via recursive feature elimination (RFE)[47]. RFE repeatedly fits a chosen regression model, and then discards a fraction of features found to be least relevant for decision making. This process is repeated until only the desired *n* descriptors remain. As the underlying regression model, we choose random forests[49–51]. A random forest is a so-called ensemble learning method, i.e., a collection of individual predictors, over which an average is calculated. This reduces overfitting and increases generalizability of the model, which is especially relevant when the training set is of limited size. The random forest consists of a number of decision trees, each of which only has access to a (randomly chosen) subset of all features for making the best possible prediction. The RFE algorithm is run 100 times with varying random seeds to counter statistical artifacts. Depending on the chosen *n*, one or more *n*-tuples of features may be selected by this process more often than other combinations. If this was the case, we picked the *n*-tuple selected most often. However, the larger *n* gets, the less likely this becomes. Thus, if this was not the case, we artificially composed the best *n*-tuple based on the frequency distribution of all descriptors included in any of the tuples selected at least once as most relevant *n*-tuple.

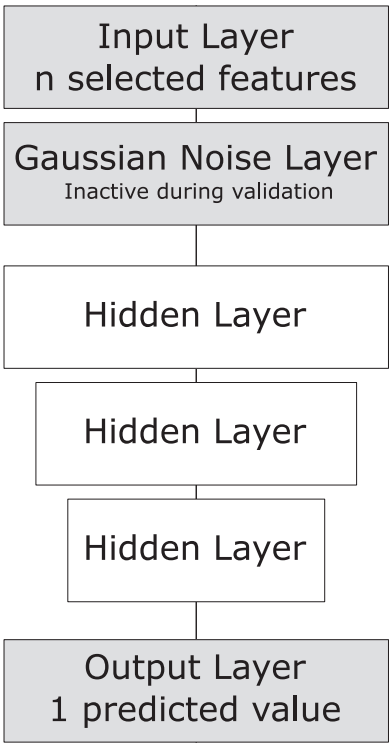


Fig. 5 Example architecture for deep learning models. General architecture of a deep learning model used for predicting corrosion inhibition efficiencies of chemical compounds (output) from molecular features (input).

Deep learning models

We evaluated the predictive value of the features identified either by *f*-test-based ANOVA or RFE by using them as input features for a deep neural network that was trained to predict the corrosion inhibition efficiency. The predictive quality of this network was then evaluated on the representative validation set withheld in the very beginning from the data (see above). Thereby we used four different types of deep neural networks: tiny models (three input features), small models (five input features), medium models (63 input features) and large models (containing the full set of 1260 available input features). Each of these models (deep neural networks) was composed by three hidden layers with a `relu` activation function (*cf.* Fig. 5). They were trained for 25 epochs using an Adam optimizer and the mean squared error (MSE) of the scaled target values as the loss function. Since the dataset was very small (only 54 training samples after withholding six samples for the representative validation set) the input data was first passed through a Gaussian noise layer with $\mu = 0$ and $\sigma = 0.1$ for each model. This layer added some Gaussian random noise in each epoch, which effectively served as a data augmentation technique and helped to improve generalization of the model and to reduce overfitting. The Gaussian noise layer was deactivated when predictions were made for the (previously unseen) validation data. The hyperparameters varied depending on the number of input parameters (model sizes) were the number of units in each hidden layer, as well as the learning rate for the Adam optimizer. For details the reader is referred to the supplementary material.

Autoencoders

Recently, autoencoders have attracted substantial attention in dimensionality reduction in the context of deep learning[69–71]. Autoencoders are however not used for predictions. Rather their objective is to generate an approximation of the input data as close as possible to themselves after compressing them through a bottleneck. Autoencoders consist of three parts: an encoder that learns how to distill the most relevant information from the input; the code, i.e., the condensed information gained from the input; and lastly the decoder, which learns how to re-construct the input data as accurately as possible from the code (*cf.* Fig. 6).

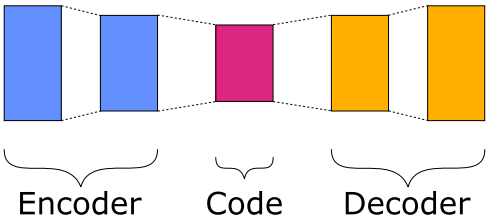


Fig. 6 Example architecture for autoencoders. Schematic illustration of an autoencoder. Each bar represents a dense layer, bars of same size indicate same number of neurons in the respective layers.

As one has substantial freedom in choosing the dimension of the code, one case use autoencoders to reduce, e.g., the 1260 input features in our problem to e.g., 2–5 key variables. Of course, the lower the dimension of the code, i.e., the greater the compression of the input data, the greater the reconstruction error typically becomes. Note that while autoencoders are quite a powerful tool for dimensionality reduction, they are not a feature selection method in the classical sense[71]. Rather they are similar to principal component analysis (PCA) which can also be used for dimensionality reduction. Neither the code produced by the autoencoder nor the principal components found by PCA have a direct physical correspondence to any of the input features. Instead, PCA constructs a linear projection of the input data onto a basis of the closest lower rank representation of the original data space. In general, a unique inversion of this process does not exist. Similarly, autoencoders typically learn a highly nonlinear mapping which approximates a bijection between the original and the latent data dimensions up to the reconstruction error[72]. The great advantage of autoencoders compared to PCA is that the decoder part can thus be used for predictions on generic data reconstructed from the latent space. We trained an autoencoder with a code of dimension 2, which was suitable to plot a two-dimensional representation of the chosen number of input features (for hyperparameters *cf.* Supplementary Table 1). Moreover, its decoder was able to map any point in this two-dimensional reduced feature space to a predicted corrosion inhibition efficiency of ZE41 (*cf.* Fig. 4). Note that besides providing a low-dimensional representation of the input data, autoencoders can also be used to reduce noise within a dataset, or to detect potential anomalies in the data[70].

DATA AVAILABILITY

The data used for this study is available at https://doi.org/10.5281/zenodo.5564824.

CODE AVAILABILITY

The code used for this study is available at https://doi.org/10.5281/zenodo.5564824.

Received: 1 June 2021; Accepted: 21 October 2021;
Published online: 01 December 2021

REFERENCES

1. Anderson, D. L. Chemical composition of the Mantle. *J. Geophys. Res.* **88 Suppl**, 41–52 (1983).
2. Taub, A. I. & Luo, A. A. Advanced lightweight materials and manufacturing processes for automotive applications. *MRS Bull.* **40**, 1045–1053 (2015).
3. Joost, W. J. & Krajewski, P. E. Towards magnesium alloys for high-volume automotive applications. *Scr. Mater.* **128**, 107–112 (2017).
4. Dziubińska, A., Gontarz, A., Dziubiński, M. & Barszcz, M. The forming of magnesium alloy forgings for aircraft and automotive applications. *Adv. Sci. Tech.* **10**, 158–168 (2016).
5. Luthringer, B. J. C., Feyerabend, F. & Willumeit-Römer, R. Magnesium-based implants: a mini-review. *Magnes. Res.* **27**, 142–54 (2014).
6. Brar, H. S., Platt, M. O., Sarntinoranont, M., Martin, P. I. & Manuel, M. V. Magnesium as a biodegradable and bioabsorbable material for medical implants. *Jom* **61**, 31–34 (2009).
7. Deng, M. et al. Ca/In micro alloying as a novel strategy to simultaneously enhance power and energy density of primary Mg-air batteries from anode aspect. *J. Power Sources* **472**, 228528 (2020).
8. Zhang, T., Tao, Z. & Chen, J. Magnesium-air batteries: from principle to application. *Mater. Horiz.* **1**, 196–206 (2014).

9. Meeusen, M. et al. A complementary electrochemical approach for time-resolved evaluation of corrosion inhibitor performance. *J. Electrochem. Soc.* **166**, C3220–C3232 (2019).
10. Muster, T. H. et al. A rapid screening multi-electrode method for the evaluation of corrosion inhibitors. *Electrochim. Acta* **54**, 3402–3411 (2009).
11. White, P. A. et al. A new high-throughput method for corrosion testing. *Corros. Sci.* **58**, 327–331 (2012).
12. White, P. A. et al. Towards materials discovery: assays for screening and study of chemical interactions of novel corrosion inhibitors in solution and coatings. *N. J. Chem.* **44**, 7647–7658 (2020).
13. Chen, F. F. et al. Correlation between molecular features and electrochemical properties using an artificial neural network. *Mater. Des.* **112**, 410–418 (2016).
14. Meftahi, N. et al. Machine learning property prediction for organic photovoltaic devices. *npj Comput. Mater.* **6**, 1–8 (2020).
15. Winkler, D. A. et al. Towards chromate-free corrosion inhibitors: structure-property models for organic alternatives. *Green. Chem.* **16**, 3349–3357 (2014).
16. Le, T. C. & Winkler, D. A. Discovery and optimization of materials using evolutionary approaches. *Chem. Rev.* **116**, 6107–6132 (2016).
17. Galvão, T. L., Novell-Leruth, G., Kuznetsova, A., Tedim, J. & Gomes, J. R. Elucidating structure–property relationships in aluminum alloy corrosion inhibitors by machine learning. *J. Phys. Chem. C* **124**, 5624–5635 (2020).
18. Feiler, C. et al. In silico screening of modulators of magnesium dissolution. *Corros. Sci.* **163**, 108245 (2020).
19. Würger, T. et al. Data science based Mg corrosion engineering. *Front. Mater.* **6**, 53 (2019).
20. Würger, T. et al. Exploring structure–property relationships in magnesium dissolution modulators. *npj Mater. Degrad.* **5**, 2 (2021).
21. Zeller-Plumhoff, B. et al. Exploring key ionic interactions for magnesium degradation insimulated body fluid—a data-driven approach. *Corros. Sci.* **182**, 109272 (2021).
22. Yuwono, J. A., Taylor, C. D., Frankel, G. S., Birbilis, N. & Fajardo, S. Understanding the enhanced rates of hydrogen evolution on dissolving magnesium. *Electrochem. Commun.* **104**, 106482 (2019).
23. Milošev, I. et al. Editors' choice—The effect of anchor group and alkyl backbone chain on performance of organic compounds as corrosion inhibitors for aluminum investigated using an integrative experimental-modeling approach. *J. Electrochem. Soc.* **167**, 061509 (2020).
24. Würger, T., Feiler, C., Vonbun-Feldbauer, G. B., Zheludkevich, M. L. & Meißner, R. H. A first-principles analysis of the charge transfer in magnesium corrosion. *Sci. Rep.* **10**, 15006 (2020).
25. Feiler, C., Mei, D., Luthringer-Feyerabend, B., Lamaka, S. & Zheludkevich, M. Rational design of effective Mg degradation modulators. *Corrosion* **77**, 204–208 (2021).
26. Poberžnik, M. et al. DFT study of n-alkyl carboxylic acids on oxidized aluminum surfaces: from standalone molecules to self-assembled-monolayers. *Appl. Surf. Sci.* **525**, 146156 (2020).
27. Fockaert, L. et al. ATR-FTIR in Kretschmann configuration integrated with electrochemical cell as in situ interfacial sensitive tool to study corrosion inhibitors for magnesium substrates. *Electrochim. Acta* **345**, 136166 (2020).
28. Mauri, A. *Methods in Pharmacology and Toxicology*, 801–820 (Humana Press Inc., 2020).
29. Landrum, G. et al. Rdkit: open-source cheminformatics. https://www.rdkit.org/ (2016).
30. Mikulskis, P., Alexander, M. R. & Winkler, D. A. Toward interpretable machine learning models for materials discovery. *Adv. Intell. Syst.* **1**, 1900045 (2019).
31. Pérez-Sánchez, G., Galvão, T. L., Tedim, J. & Gomes, J. R. A molecular dynamics framework to explore the structure and dynamics of layered double hydroxides. *Appl. Clay Sci.* **163**, 164–177 (2018).
32. Klink, S., Höche, D., La Mantia, F. & Schuhmann, W. FEM modelling of a coaxial three-electrode test cell for electrochemical impedance spectroscopy in lithium ion batteries. *J. Power Sources* **240**, 273–280 (2013).
33. Hammerich, M. et al. Heterodiazocines: synthesis and photochromic properties, trans to cis switching within the bio-optical window. *J. Am. Chem. Soc.* **138**, 13111–13114 (2016).
34. Ma, R., Huang, D., Zhang, T. & Luo, T. Determining influential descriptors for polymer chain conformation based on empirical force-fields and molecular dynamics simulations. *Chem. Phys. Lett.* **704**, 49–54 (2018).
35. Ash, J. & Fourches, D. Characterizing the chemical space of ERK2 kinase inhibitors using descriptors computed from molecular dynamics trajectories. *J. Chem. Inf. Model.* **57**, 1286–1299 (2017).
36. Pereira, F. & Aires-de Sousa, J. Machine learning for the prediction of molecular dipole moments obtained by density functional theory. *J. Cheminform.* **10**, 43 (2018).
37. Nørskov, J. K., Abild-Pedersen, F., Studt, F. & Bligaard, T. Density functional theory in surface chemistry and catalysis. *Proc. Natl Acad. Sci.* **108**, 937–943 (2011).
38. Richert, C. & Huber, N. A review of experimentally informed micromechanical modeling of nanoporous metals: from structural descriptors to predictive structure–property relationships. *Materials* **13**, 3307 (2020).
39. Morales-Gil, P., Walczak, M. S., Cottis, R. A., Romero, J. M. & Lindsay, R. Corrosion inhibitor binding in an acidic medium: interaction of 2-mercaptobenizmidazole with carbon-steel in hydrochloric acid. *Corros. Sci.* **85**, 109–114 (2014).
40. Winkler, D. A. et al. Using high throughput experimental data and in silico models to discover alternatives to toxic chromate corrosion inhibitors. *Corros. Sci.* **106**, 229–235 (2016).
41. Kokalj, A. et al. Simplistic correlations between molecular electronic properties and inhibition efficiencies: Do they really exist? *Corros. Sci.* **179**, 108856 (2021).
42. Johnson, K. J. & Synovec, R. E. Pattern recognition of jet fuels: comprehensive GC × GC with anova-based feature selection and principal component analysis. *Chemom. Intell. Lab.* **60**, 225–237 (2002).
43. Burgard, D. R. *Chemometrics: Chemical and Sensory Data* (CRC Press, 2018).
44. Kim, T. K. Understanding one-way anova using conceptual figures. *Korean J. Anesthesiol.* **70**, 22–26 (2017).
45. Bijma, F., Jonker, M., van der Vaart, A. & Erné, R. *An Introduction to Mathematical Statistics* (Amsterdam University Press, 2017).
46. Chandrashekar, G. & Sahin, F. A survey on feature selection methods. *Comput. Electr. Eng.* **40**, 16–28 (2014).
47. Guyon, I., Weston, J., Barnhill, S. & Vapnik, V. Gene selection for cancer classification using support vector machines. *Mach. Learn.* **46**, 389–422 (2002).
48. Solorio-Fernández, S., Carrasco-Ochoa, J. A. & Martínez-Trinidad, J. F. A review of unsupervised feature selection methods. *Artif. Intell. Rev.* **53**, 907–948 (2020).
49. Ho, T. K. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 278–282 (IEEE, 1995).
50. Genuer, R., Poggi, J.-M. & Tuleau-Malot, C. Variable selection using random forests. *Pattern Recogn. Lett.* **31**, 2225–2236 (2010).
51. Chavent, M., Genuer, R. & Saracco, J. Combining clustering of variables and feature selection using random forests. *Commun. Stat. Simul. Comput.* **50**, 426–445 (2021).
52. Eklund, M., Norinder, U., Boyer, S. & Carlsson, L. Choosing feature selection and learning algorithms in qsar. *J. Chem. Inf. Model.* **54**, 837–843 (2014).
53. Blaschke, T., Olivecrona, M., Engkvist, O., Bajorath, J. & Chen, H. Application of generative autoencoder in de novo molecular design. *Mol. Inform.* **37**, 1700123 (2018).
54. Samanta, S., O'Hagan, S., Swainston, N., Roberts, T. J. & Kell, D. B. Vae-sim: a novel molecular similarity measure based on a variational autoencoder. *Molecules* **25**, 3446 (2020).
55. Lamaka, S. V. et al. Comprehensive screening of Mg corrosion inhibitors. *Corros. Sci.* **128**, 224–240 (2017).
56. Schneider, G., Neidhart, W., Giller, T. & Schmid, G. 'Scaffold-Hopping' by topological pharmacophore search: a contribution to virtual screening. *Angew. Chem. Int. Ed.* **38**, 2894–2896 (1999).
57. Fechner, U., Franke, L., Renner, S., Schneider, P. & Schneider, G. Comparison of correlation vector methods for ligand-based similarity searching. *J. Comput. Aid. Mol. Des.* **17**, 687–698 (2003).
58. Grisoni, F., Merk, D., Byrne, R. & Schneider, G. Scaffold-hopping from synthetic drugs by holistic molecular representation. *Sci. Rep.* **8**, 1–12 (2018).
59. Labute, P. A widely applicable set of descriptors. *J. Mol. Graph. Model.* **18**, 464–477 (2000).
60. Devinyak, O., Havrylyuk, D. & Lesyk, R. 3D-MoRSE descriptors explained. *J. Mol. Graph. Model.* **54**, 194–203 (2014).
61. Schuur, J. H., Selzer, P. & Gasteiger, J. The coding of the three-dimensional structure of molecules by molecular transforms and its application to structure-spectra correlations and studies of biological activity. *J. Chem. Inf. Comp. Sci.* **36**, 334–344 (1996).
62. Dean, J. A. *Lange's Chemistry Handbook*. (University of Tennessee, McGrawHill, Inc, 1999).
63. Smith, R. & Martell, A. *Critical Stability Constants, Vol. 3. Other Organic Ligands*, vol. 365 (Plenum Press, 1977).
64. David, L., Thakkar, A., Mercado, R. & Engkvist, O. Molecular representations in AI-driven drug discovery: a review and practical guide. *J. Cheminform.* **12**, 1–22 (2020).
65. TURBOMOLE. *V7.4. A Development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2019 since 2007* (TURBOMOLE GmbH, 2019).
66. Staroverov, V. N., Scuseria, G. E., Tao, J. & Perdew, J. P. Comparative assessment of a new nonempirical density functional: molecules and hydrogen-bonded complexes. *J. Chem. Phys.* **119**, 12129–12137 (2003).
67. Eichkorn, K., Weigend, F., Treutler, O. & Ahlrichs, R. Auxiliary basis sets for main row atoms and transition metals and their use to approximate coulomb potentials. *Theor. Chem. Acc.* **97**, 119–124 (1997).
68. Pedregosa, F. et al. Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).

69. Wang, Y., Yao, H. & Zhao, S. Auto-encoder based dimensionality reduction. *Neurocomputing* **184**, 232–242 (2016).
70. Sakurada, M. & Yairi, T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* 4–11 (ACM, 2014).
71. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).
72. Almotiri, J., Elleithy, K. & Elleithy, A. Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)* 1–5 (IEEE, 2017).

ACKNOWLEDGEMENTS

Funding by the Helmholtz Association is gratefully acknowledged. T.W. and C.F. gratefully acknowledge funding by the Deutscher Akademischer Austauschdienst (DAAD, German Academic Exchange Service) via Projektnummer 57511455. R.M. gratefully acknowledges funding by the Deutsche Forschungsgemeinschaft (D.F.G., German Research Foundation) via Projektnummer 192346071-SFB 986 and Projektnummer 390794421-GRK 2462.

AUTHOR CONTRIBUTIONS

E.J.S., T.W., S.V.L., R.H.M., C.J.C., M.L.Z., C.F., and R.C.A. contributed to the conception and design of the study. C.F. generated the molecular descriptor database. E.J.S. did the theoretical analyses and wrote the supporting code. E.J.S., T.W., R.C.A., and C.F. evaluated the quality of the presented models. E.J.S. and T.W. created the figures. E.J.S. and C.F. wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

FUNDING

Open Access funding enabled and organized by Projekt DEAL.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41524-021-00658-7.

Correspondence and requests for materials should be addressed to Christian Feiler or Roland C. Aydin.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Predicting the Inhibition Efficiencies of Magnesium Dissolution Modulators using Sparse Machine Learning Models

SUPPLEMENTARY MATERIAL

Elisabeth J. Schiessler[1], Tim Würger[2, 3], Sviatlana V. Lamaka[2], Robert H. Meißner[2], Christian J. Cyron[1], Mikhail L. Zheludkevich[2, 4], Christian Feiler✉, [2], and Roland C. Aydin✉, [1]

[1]Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Geesthacht, Germany
[2]Institute of Surface Science, Helmholtz-Zentrum Hereon, Geesthacht, Germany
[3]Institute of Polymers and Composites, Hamburg University of Technology, Hamburg, Germany
[4]Institute for Materials Science, Faculty of Engineering, Kiel University, Kiel, Germany

SUPPLEMENTARY NOTES

Probability of Selecting Specific Features

We want to calculate the probability of a random 5-tuple of features containing any specific molecular descriptor. Since the tuple is not ordered, the number of possible combinations of k-tuples is given by the formula for drawing k elements out of a set of N samples via the binomial coefficient

$$\binom{N}{k} = \frac{N!}{k! \cdot (N-k)!}. \tag{1}$$

Thus, the number of possible 5-tuples equals

$$\binom{1260}{5} \approx 2.63 \cdot 10^{13}, \tag{2}$$

whereas the number of 5-tuples containing a specific element is given by

$$\binom{1259}{4} \approx 1.04 \cdot 10^{11}. \tag{3}$$

The probability that a specific molecular descriptor is included in a randomly selected 5-tuple thus equals

$$\frac{1.04 \cdot 10^{11}}{2.63 \cdot 10^{13}} \approx 3.97 \cdot 10^{-3} \approx 0.004 \tag{4}$$

We performed a 5-fold cross-validation on selecting 5-tuples, with 100 runs each, so 500 runs in total. Thus, the expected value for a specific feature to be included would be 1.98 runs. The descriptor *LUMO / eV* was included in 479/500 runs, which is a factor of $2.41 \cdot 10^2$ more often than the expected value.

SUPPLEMENTARY TABLES

Model Hyperparameters

The regression models all contain 3 hidden layers that use `relu` activations. We choose an Adam optimizer and use mean squared error (MSE) as the loss function. Layer sizes vary, as do learning rates, *cf.* Supplementary Table 1. The models all contain a Gaussian noise layer with $\mu = 0$ and $\sigma = 0.1$ that is only active during training, and are trained for 25 epochs each.

The autoencoder contains three hidden layers with `leaky relu` activations, and does not use a Gaussian noise layer. It is compiled using an Adam optimizer with learning rate of 0.01 and MSE as loss function. The model is trained for 100 epochs.

Supplementary Table 1: Chosen hyperparameters for the different model types.

Model category	Layer sizes	Learning rate	Task
tiny	50-20-10	0.01	Regression
small	50-20-10	0.01	Regression
medium	50-20-10	0.05	Regression
large	100-50-10	0.005	Regression
small	10-2-10	0.01	Autoencoder

Top 63 Molecular Descriptors

The top 63 features as identified by analysis of variance (ANOVA) are provided in Supplementary Table 2. The top 63 features as identified by recursive feature elimination (RFE) are provided in Supplementary Table 3.

Mean Model Performance across all 10 Cross Validation Folds

The mean values of all reported statistics over all 10 cross validation folds, feature selection and model types can be found in Supplementary Table 4.

Model Performance for the Representative Validation Set

The average and standard deviations of predictions on the representative validation set from 100 training runs over all feature selection methods and model types can be found in Supplementary Table 5. Summarising statistics are provided in Supplementary Table 6.

Experimental Data

More information on the investigated compounds, including their SMILES strings and experimentally determined inhibition efficiencies (IE) for ZE41, can be found in Supplementary Table 7.

SUPPLEMENTARY REFERENCES

[1] Lamaka, S. V. *et al.* Comprehensive screening of Mg corrosion inhibitors. *Corros. Sci.* **128**, 224–240 (2017).

✉ email: christian.feiler@hereon.de, roland.aydin@hereon.de

Supplementary Table 2: Top 63 most relevant features as determined by ANOVA.

Rank	Feature	Rank	Feature	Rank	Feature	Rank	Feature
1	CATS2D_03_AP	17	Mor19m	33	F03[N-O]	49	Mor12p
2	CATS3D_03_AP	18	Mv	34	SsNH2	50	Eta_FL_A
3	CATS3D_02_AP	19	Mor12e	35	CATS3D_04_NL	51	GATS1m
4	LUMO / eV	20	B02[C-N]	36	E3p	52	Ui
5	P_VSA_MR_5	21	Eta_F_A	37	Hlgap / eV	53	Mor12m
6	P_VSA_LogP_2	22	R2e+	38	CATS2D_04_AA	54	SHED_PN
7	CATS2D_02_PN	23	Mor14s	39	Mor08i	55	SHED_DP
8	CATS2D_03_DP	24	NssCH2	40	HOMO / eV	56	GATS1v
9	H3m	25	Mor32m	41	TDB03e	57	MLOGP
10	nRNH2	26	SHED_AP	42	TDB02e	58	MPC07
11	H%	27	P_VSA_ppp_con	43	B03[N-O]	59	nN
12	Mor14u	28	Mor14v	44	CATS3D_02_AN	60	DLS_05
13	CATS2D_03_AN	29	Mor08s	45	Mi	61	nArCOOH
14	Eta_epsi_5	30	E2m	46	Mor27m	62	Mor08p
15	CATS3D_03_DP	31	SPH	47	TDB04v	63	SpPosA_B(e)
16	P_VSA_e_3	32	MATS7m	48	ZM2V		

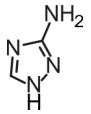
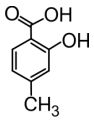
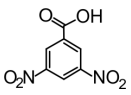
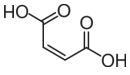
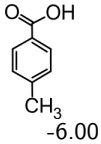
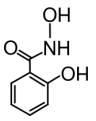
Supplementary Table 3: Top 63 most relevant features as determined by RFE.

Rank	Feature	Rank	Feature	Rank	Feature	Rank	Feature
1	P_VSA_MR_5	17	H3m	33	VE2sign_G	49	E3e
2	Mor22s	18	TDB04s	34	SpMAD_RG	50	Mor13u
3	Mor04m	19	E2s	35	R3s+	51	Eig03_AEA(dm)
4	LUMO / eV	20	R5p+	36	R5e+	52	X4Av
5	E1p	21	R2e+	37	E2v	53	P_VSA_e_3
6	HOMO / eV	22	ISH	38	Mor15i	54	Mor29e
7	P_VSA_LogP_2	23	DISPm	39	T(N..O)	55	Mor16m
8	Mor29v	24	R5i+	40	R2u+	56	GATS5m
9	MATS5v	25	Mor04i	41	MATS8p	57	E3p
10	Mor14s	26	Ds	42	Eta_epsi_5	58	E2e
11	Mor14u	27	Mor03s	43	MATS4s	59	X3Av
12	CATS3D_02_AP	28	E2m	44	H0v	60	Mor19u
13	GATS5v	29	Mor28s	45	Hy	61	GATS4s
14	MATS5m	30	Mor11u	46	E1i	62	E3v
15	GATS2s	31	TDB03m	47	VE1sign_G	63	TDB04m
16	Mor32m	32	Mor19m	48	Mor15s		

Supplementary Table 4: Mean values of root mean squared errors (RMSE), coefficients of determination (R^2), correlation coefficients (Pearson's r) and p-values of the full 10-fold cross validation for all trained models by model type and feature selection method (a: ANOVA, b: RFE, c: random selection).

No. of features	3			5			63			1260
Model Type	tiny model			small model			medium model			large model
Selection Method	a	b	c	a	b	c	a	b	c	
RMSE / pp	70	65	78	61	63	75	61	63	70	72
R^2	0.40	0.47	0.42	0.56	0.55	0.35	0.56	0.62	0.47	0.44
Pearson's r	0.53	0.65	0.53	0.69	0.72	0.50	0.73	0.77	0.63	0.61
p-value	0.38	0.20	0.30	0.20	0.14	0.37	0.14	0.10	0.24	0.26

Supplementary Table 5: Predictions for all trained models by type for the representative validation set. The averages and standard deviations are calculated over 100 training runs.

Model	Validation inhibition efficiency / %					
Index	0	5	13	36	45	54
						
True Value	-157.00	39.00	38.00	12.00	-6.00	-17.00
M3a	-105.01	-34.98	148.79	17.64	-69.53	-29.09
	± 35.58	± 22.34	± 50.81	± 24.70	± 22.45	± 23.01
M5a	-100.74	-35.98	154.62	15.88	-69.72	-30.13
	± 39.29	± 20.27	± 56.17	± 23.81	± 22.98	± 21.06
M63a	-104.19	-33.72	149.63	14.24	-69.31	-28.54
	± 35.85	± 22.21	± 57.00	± 21.76	± 24.12	± 23.43
M3b	-140.43	63.30	147.96	34.83	-22.28	19.11
	± 33.15	± 18.21	± 52.35	± 28.93	± 20.72	± 27.66
M5b	-138.90	65.77	144.03	32.88	-19.88	23.68
	± 33.19	± 19.36	± 49.41	± 29.44	± 22.36	± 31.75
M63b	-142.42	64.90	146.93	30.14	-20.63	18.86
	± 32.52	± 18.40	± 53.77	± 27.20	± 19.22	± 27.80
M3c	-36.90	-25.75	8.70	-33.71	-33.64	-29.09
	± 68.92	± 28.62	± 85.66	± 31.67	± 27.23	± 28.77
M5c	-41.96	-16.33	28.68	-23.96	-26.99	-27.66
	± 48.06	± 30.42	± 105.86	± 43.23	± 32.82	± 37.10
M63c	-73.99	17.67	127.29	10.09	-21.94	-12.83
	± 48.54	± 37.49	± 96.01	± 52.30	± 41.77	± 44.33
M1260	-77.54	14.46	64.30	8.63	-19.81	-4.01
	± 48.67	± 41.44	± 97.76	± 46.00	± 37.87	± 39.35

Supplementary Table 6: Coefficients of determination (R^2), root mean squared errors (RMSE), correlation coefficients (Pearson's r) and p-values of the full validation set predictions for all trained models by model type and feature selection method (a: ANOVA, b: RFE, c: random selection).

No. of features	3			5			63			1260
Model Type	tiny model			small model			medium model			large model
Selection Method	a	b	c	a	b	c	a	b	c	
RMSE / pp	71	50	61	67	49	55	64	49	51	36
R^2	0.26	0.83	0.28	0.38	0.84	0.42	0.41	0.84	0.56	0.81
Pearson's r	0.51	0.91	0.53	0.62	0.91	0.65	0.63	0.91	0.75	0.90
p-value	0.30	0.01	0.28	0.19	0.01	0.17	0.17	0.01	0.09	0.02

Supplementary Table 7: Investigated compounds with their SMILES strings and experimentally determined inhibition efficiencies (IE) for ZE41. Presented values of IE were determined by comparing the amount of evolved hydrogen during immersion of ZE41 alloy chips in pure 0.5% NaCl solution with the amount of hydrogen that evolves in presence of one of the listed dissolution modulators at an operation concentration of 0.05 M. The elemental composition of the alloy is provided within the published database of inhibition efficiencies [1].

Compound	SMILES	IE / %
3-Amino-1,2,4-triazole	c1[nH]nc(n1)N	-157
3-Methylcatechol	c1(c(c(ccc1)O)O)C	-31
3-Methylsalicylic acid	c1(c(c(ccc1)C(=O)O)O)C	75
4-Aminosalicylic acid	c1c(c(ccc1N)C(=O)O)O	57
4-Hydroxybenzoic acid	c1c(ccc(c1)O)C(=O)O	-170
4-Methylsalicylic acid	c1c(c(ccc1C)C(=O)O)O	39
5-Aminosalicylic acid	c1c(c(cc(c1)N)C(=O)O)O	66
5-Methylsalicylic acid	c1c(c(cc(c1)C)C(=O)O)O	55
5-Nitrobarbituric acid	C1(=O)C(C(=O)NC(=O)N1)N(=O)=O	51
2,3-Pyridinedicarboxylic acid	c1(c(cccn1)C(=O)O)C(=O)O	32
2,5-Pyridinedicarboxylic acid	c1(ccc(C(=O)O)cn1)C(=O)O	52
3,4-Dihydroxybenzoic acid	c1c(ccc(c1O)O)C(=O)O	-270
3,5-Dimethylpyrazole	c1(cc(n[nH]1)C)C	-67
3,5-Dinitrosalicylic acid	c1(c(c(cc(c1)N(=O)=O)C(=O)O)O)N(=O)=O	38
5,5-Dimethylhydation	C1(CNC(=O)N1)(C)C	-172
Ascorbic acid	[C@@H]1(OC(=O)C(=C1O)O)[C@H](CO)O	6
Asparagine	[C@H](CC(=O)N)(C(=O)O)N	-188
Aspartic acid	[C@H](CC(=O)O)(C(=O)O)N	-54
Benzoic acid	c1cc(ccc1)C(=O)O	34
Benzotriazole	c1ccc2c(c1)nn[nH]2	-108
Bicine	N(CC(=O)O)(CCO)CCO	-111
Bipyridine	c1(ccccn1)c1ccccn1	6
Bismuthiol	c1(nnc(s1)S)S	-17
Citric acid	C(C(CC(=O)O)(C(=O)O)O)C(=O)O	-47
Cystein	[C@H](CS)(C(=O)O)N	-104
Diethylentriamine	C(CNCCN)N	-18
Diglycolic acid	C(=O)(O)COCC(=O)O	60
Formic acid	C(=O)O	-5
Fumaric acid	C(=C\\C(=O)O)/C(=O)O	17
Gluconic acid	[C@H]([C@@H]([C@H]([C@H](CO)O)O)O)(C(=O)O)O	48
Glutamic acid	[C@H](CCC(=O)O)(C(=O)O)N	-139
Glycerol	C(C(CO)O)O	-13
Glycine	C(C(=O)O)N	-215
Glycolic acid	C(=O)(C)CO	59
Kojic acid	c1(cc(=O)c(co1)O)CO	45
Lysine	[C@H](CCCCN)(C(=O)O)N	-113
Maleic acid	C(=C\\C(=O)O)\\C(=O)O	12
Maltol	c1(c(c(=O)cco1)O)C	9
Mandelic acid	c1ccc(cc1)[C@@H](C(=O)O)O	-6
Nicotinic acid	c1c(cccn1)C(=O)O	3
Norleucine	CCC[C@@H](C(=O)O)N	-138
NTA	N(CC(=O)O)(CC(=O)O)CC(=O)O	-124
Oxalic acid	C(=O)(O)C(=O)O	52
Panthenol	C(=O)(NCCCO)[C@H](C(C)(CO)C)O	-98
p-^{t}Bu-Benzoic acid	c1c(ccc(c1)C(C)(C)C)C(=O)O	31
p-Toluic acid	c1c(ccc(c1)C)C(=O)O	-6
Phenylalanine	[C@H](Cc1ccccc1)(C(=O)O)N	-146
Phthalic acid	c1c(c(ccc1)C(=O)O)C(=O)O	31
Picolinic acid	c1(ccccn1)C(=O)O	54
Piperazine	C1CNCCN1	-145
Pyrazole	c1ccn[nH]1	-38
Quinaldic acid	c1(ccc2c(n1)cccc2)C(=O)O	21
Quinic acid	[C@@]1(CC[C@H]([C@@H]([C@@H]1O)O)O)(O)C(=O)O	19
Salicylaldoxime	c1c(c(ccc1)/C=N/O)O	-89
Salicylhydroxamic acid	c1c(c(ccc1)C(=O)NO)O	-17
Salicylic acid	c1c(c(ccc1)C(=O)O)O	37
Tartaric acid	[C@H]([C@H](C(=O)O)O)(C(=O)O)O	46
Tris	C(CO)(CO)(CO)N	-194
Uracil	[nH]1ccc(=O)[nH]c1=O	-108

A.3 Publication 3

Searching the chemical space for effective magnesium dissolution modulators: A deep learning approach using sparse features

E. J. Schiessler, T. Würger, B. Vaghefinazari, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler and R. C. Aydin

E.J.S., T.W., B.V., S.V.L., R.H.M., C.J.C., M.L.Z., C.F. and R.C.A. contributed to the conception and design of the study. C.F. and T.W. generated the molecular descriptor database and selected the compounds for experimental validation of the model. B.V and S.V.L. conducted the validation experiments. E.J.S. did the theoretical analyses and wrote the supporting code. E.J.S., T.W., R.C.A and C.F. evaluated the quality of the presented models. E.J.S. and T.W. created the figures. E.J.S., T.W., C.F. and R.C.A. wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

Reprinted from E. J. Schiessler, T. Würger, B. Vaghefinazari, S. V. Lamaka, R. H. Meißner, C. J. Cyron, M. L. Zheludkevich, C. Feiler, and R. C. Aydin. Searching the Chemical Space for Effective Magnesium Dissolution Modulators: A Deep Learning Approach using Sparse Features. *npj Materials Degradation*, 7:74, September 2023. 10.1038/s41529-023-00391-0, *licensed under the Creative Commons Attribution 4.0 License* (http://creativecommons.org/licenses/by/4.0/).

ARTICLE OPEN

Check for updates

Searching the chemical space for effective magnesium dissolution modulators: a deep learning approach using sparse features

Elisabeth J. Schiessler[1], Tim Würger[2,3], Bahram Vaghefinazari[2], Sviatlana V. Lamaka[2], Robert H. Meißner[2,3], Christian J. Cyron[1,4], Mikhail L. Zheludkevich[2,5,6], Christian Feiler[2,6]✉ and Roland C. Aydin[1,4]✉

Small organic molecules can alter the degradation rates of the magnesium alloy ZE41. However, identifying suitable candidate compounds from the vast chemical space requires sophisticated tools. The information contained in only a few molecular descriptors derived from recursive feature elimination was previously shown to hold the potential for determining such candidates using deep neural networks. We evaluate the capability of these networks to generalise by blind testing them on 15 randomly selected, completely unseen compounds. We find that their generalisation ability is still somewhat limited, most likely due to the relatively small amount of available training data. However, we demonstrate that our approach is scalable; meaning deficiencies caused by data limitations can presumably be overcome as the data availability increases. Finally, we illustrate the influence and importance of well-chosen descriptors towards the predictive power of deep neural networks.

npj Materials Degradation (2023)7:74; https://doi.org/10.1038/s41529-023-00391-0

INTRODUCTION

Magnesium (Mg) and its alloys have distinct properties that render them promising materials for various applications, ranging from aerospace and automotive to biomedical and energy storage. However, it is essential to control the surface reactivity characteristics of Mg to unlock its full potential in each particular application field. For example, preventing corrosion is crucial for transport applications (e.g., aerospace and automotive), while medical applications (e.g., temporary biodegradable implants) require tailored degradation rates. For batteries with a Mg anode, the dissolution rate has to be adapted to maintain a constant output voltage and to protect the utilisation efficiency, e.g., from the occurring chunk effect[1–3]. Small organic molecules exhibit great potential in controlling corrosion in these applications, for which they are—depending on the target application— typically incorporated into a complex coating system in transportation applications or become a dissolved component of the electrolyte in Mg-air batteries.

The chemical space of compounds with potentially useful properties is practically infinite [4], rendering purely experimental approaches insufficient despite impressive progress in the field of high-throughput testing. Data-driven computational methods have emerged as powerful tools for the prediction and identification of useful corrosion inhibitors and can thus enable a more efficient design of experiments. Exploring large areas of chemical space can become orders of magnitude faster, allowing the preselection of promising candidates for in-depth experimental testing. At the same time, further insights into the underlying chemical mechanisms of corrosion and its inhibition can be obtained, which in turn provide additional input features for predictive quantitative structure-property relationships (QSPRs).

To develop accurate and robust predictive models, a sufficiently large, reliable, and chemically diverse database is required, reflecting the complexity of the relevant chemical environment. Cheminformatics software packages, such as RDKit and alvaDesc, enable the structural encoding of the numerous different functional entities and molecular features included in such databases. Aside from that, advances in computing power and simulation algorithms have enabled simulations (e.g., relying on density functional theory or (semi empirical) force field calculations) that can provide a wide range of potentially useful molecular descriptors[5]. By selecting only the most suitable descriptors and using them as input for a QSPR model, a more thorough and nuanced analysis of the potential effectiveness of a given compound can be provided. As additional data becomes available, the model can be continually refined and improved, ensuring that the most effective dissolution modulators are identified.

The predictive performance of the trained QSPR model depends significantly on the selected molecular features, as high correlation between input features or low correlation with the target property can compromise the model. In recent years, machine learning models have become increasingly popular in corrosion modelling[6–9]. In Schiessler et al. [10], we compared the capabilities of statistical methods, such as the analysis of variance (ANOVA[11–14]), with recursive feature elimination (RFE[15]) based on random forests[16–18] in selecting suitable input features of 60 compounds for a deep neural network to predict the corrosion inhibition efficiencies of chemical compounds for the magnesium alloy ZE41. Descriptors derived from density functional theory calculations could be identified as highly significant for predicting the experimental performance of corrosion inhibitors, when joined with input features derived from the molecular structure.

[1]Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Geesthacht, Germany. [2]Institute of Surface Science, Helmholtz-Zentrum Hereon, Geesthacht, Germany. [3]Institute of Polymers and Composites, Hamburg University of Technology, Hamburg, Germany. [4]Institute for Continuum and Material Mechanics, Hamburg University of Technology, Hamburg, Germany. [5]Institute for Materials Science, Faculty of Engineering, Kiel University, Kiel, Germany. [6]Kiel Nano, Surface and Interface Science KiNSIS, Kiel University, Kiel, Germany. ✉email: christian.feiler@hereon.de; roland.aydin@hereon.de

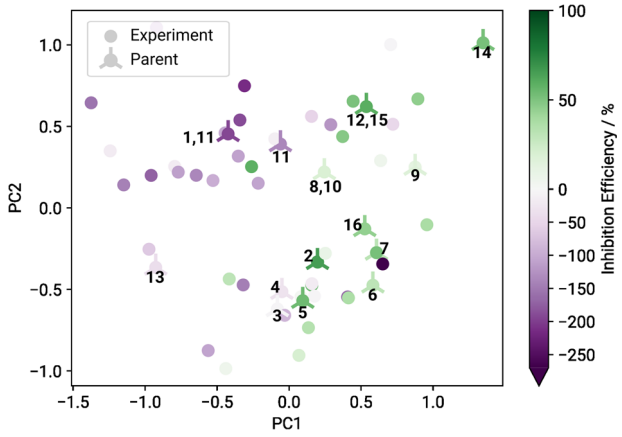


Fig. 1 Structure-property landscape of 60 magnesium dissolution modulators for the magnesium-based alloy ZE41. The axes represent the two principal components (PC) resulting from the kernel principal component analysis. Based on this map, untested compounds of interest were selected for further investigation using the ExChem routine. Twenty of the original 60 structures were randomly chosen as `parents' (crossed circles), for which highly similar compounds were determined out of a pool of commercially available chemicals. The numbers indicate which of the selected test candidates as defined in Table 1 correspond to which parents.

Combining the sparse feature selection strategies with deep learning forms a predictive QSPR framework that can be used for the identification of promising corrosion inhibitors. However, when working with small datasets there exists a risk of overfitting on the training data, which will lead to results that do not generalise well and may not be able to give useful insights beyond the training domain[19,20].

In this study, we predict and test the corrosion inhibition efficiencies of 15 previously unseen compounds that were selected using the ExChem[21] routine to evaluate the limitations of the models presented in Schiessler et al.[10]. The fundamental concept of ExChem is based on molecular similarities calculated from the Smooth Overlap of Atomic Positions (SOAP)[22,23] approach. The molecules in the dataset that was used to train the underlying supervised machine learning model are represented in form of a 2D map following a dimension reduction approach, thereby visualising the relationships between molecular structure and corrosion inhibition performance via the formation of similarity clusters. Moreover, ExChem facilitates the projection of a database of commercially available compounds onto the landscape of known chemical space and thus enables a rational selection of compounds for subsequent experimental evaluation based on structural similarities between the two databases and by providing estimates for the corrosion inhibition performance of the untested small organic molecules. After confirming the robustness of the feature selection process, the predictive performance of the neural networks is evaluated. Identified outliers are discussed with respect to their chemical features to explain deviations occurring between experimental and predicted corrosion inhibition properties. Furthermore we assess the effect of integrating more data into the training set and confirm the scalability of our approach.

RESULTS AND DISCUSSION

Similarity-based compound selection

Under the overarching goal to find promising magnesium dissolution modulators for the magnesium alloy ZE41 in the vast chemical space, we tested the limits of the machine learning models as presented in our previous study[10] with respect to prediction performance and scalability. Therefore, we selected blind test candidates using the ExChem routine from a database of over 7000 commercially available chemicals, as provided by Thermo Fisher Scientific[21]. A database of 60 magnesium dissolution modulators for ZE41, originally used to train the machine learning models, served as foundation for the approach[10,24]. Molecular similarities of the original training data and the database of commercially available compounds were calculated using the SOAP kernel with a cutoff radius $r_c = 2.0$ Å, a Gaussian width $\xi = 0.3$ Å and $\zeta = 2$ (cf. Methods)[22,23]. We reduced the resulting high-dimensional similarity matrix to two dimensions using kernel principal component analysis. Correlating the two-dimensional data with experimentally measured corrosion inhibition efficiencies for the respective compounds resulted in a structure-property landscape, as shown in Fig. 1.

A clear relationship between molecular structure and corrosion inhibition efficiency becomes evident, where compounds yielding corrosion inhibiting effects are located predominantly on the right side of the landscape (green circles) and compounds accelerating corrosion are located mainly on the left side (purple circles). The ExChem routine was used to identify potential test candidates in the commercial database that exhibit high similarity to certain compounds that were already experimentally validated. Initially, 20 compounds of interest were randomly selected from the experimental database. Each compound served as reference ('parent') to identify five highly similar structures ('children') in the commercial database based on the underlying SOAP similarities. Out of the resulting 100 structures, 20 were randomly chosen for experimental blind testing. Since four of these 20 were not soluble in water, they were removed from the pool of blind test candidates. The remaining 16 selected compounds are listed in Table 1 along with their respective indices, names and experimentally measured inhibition efficiencies. The associated parent structures are marked with crossed circles in Fig. 1 along with the indices of the selected children, i.e., the chosen blind test candidates. Compound 2 was excluded during the evaluation phase as the required materials could not be delivered. In the following, we evaluate the robustness of the feature selection process given the availability of this additional dataset, as well as the performance of the predictive models against the presented blind test data, which have been withheld from the model training process.

Feature selection robustness

We investigated the quality of selected features that were presented in our previous study[10] by exploring how susceptible the feature selection results are to changes in input data. The original 60 sample dataset[10,24] was augmented by the 15 blind testing samples given in Table 1, forming a combined dataset of 75 compounds. This gave us a number of dataset compositions that we use throughout this manuscript:

- *original* dataset (60 compounds): DS_{60}
- *blind testing* dataset (15 compounds): DS_{15}
- *combined* dataset (75 compounds): DS_{75}

On each composition, we performed grouped feature selection using RFE based on random forests. Data were split into 10 cross-validation folds (which differ per dataset composition), and on each fold the process was repeated 100 times using varying random seeds. From the resulting 1,000 top five groups per dataset composition we report the ones that got selected most often, cf. Table 2.

As we can see in Table 2, the top five feature sets FS_{60} and FS_{75} found for the original (DS_{60}) and combined (DS_{75}) dataset compositions overlap in three out of five components. The remaining two from each set (CATS3D_02_AP and Mor04m for FS_{60}, HOMO and E2s for FS_{75}) do in fact come up in the other dataset composition's respective best feature sets list, just not in

Table 1. Blind testing compounds.

Index	Compound	IE ZE41 / %
1	2-Amino-2-methyl-1,3-propanediol	−152 ± 6
~~2~~	~~3-Aminophthalic acid~~	–
3	3-Hydroxyacetophenone	−33 ± 4
4	4-Hydroxybenzylalcohol	−135 ± 3
5	4-Pyridinecarboxylic acid	40 ± 2
6	2,4-Dihydroxybenzoic acid	−141 ± 2
7	3,4-Pyridinedicarboxylic acid	76 ± 1
8	D-Glucose	−23 ± 2
9	Itaconic acid	64 ± 3
10	L-Sorbose	46 ± 2
11	L-Threonine	−216 ± 10
12	Methylmalonic acid	35 ± 3
13	Pyridazine	16 ± 1
14	Tartronic acid	−33 ± 4
15	Tricarballylic acid	50 ± 17
16	Vanillic acid	−119 ± 13

Experimentally measured corrosion inhibition efficiencies for the selected blind testing compounds. Compound 2 was excluded during the evaluation phase as the required materials could not be delivered. The initial pH of all tested compounds was adjusted to 7.0 ± 0.1 by NaOH solution.

first place. FS_{60} and FS_{75} were chosen in 38% and 30% of cases respectively. The winning feature set FS_{15} for the blind testing dataset composition DS_{15} on the other hand was chosen in only 12% of all runs, with a greater variation in included candidates. This comes as no surprise, as 15 data points is quite few in most machine learning contexts. The best features for the original dataset, FS_{60}, have no overlap with the blind testing set winners FS_{15}. From this we surmise a somewhat limited ability of FS_{60} to accurately capture the specific properties of the blind testing dataset, as well as a reduced capacity to generalise. The winning feature set FS_{75} determined from the combined dataset composition includes descriptors from both FS_{60} and FS_{15}. It is noteworthy that HOMO, a DFT-derived descriptor denoting the highest occupied molecular orbital energy level, was present in the second best feature set for DS_{60}, and came up in the shared first place for best feature set in our original study[10]. This descriptor is included in both FS_{75} and FS_{15} and seems to play a crucial role in capturing properties of the presented corrosion inhibition dataset.

Feature selection robustness was furthermore investigated under change of target metric (using inhibition power/dB instead of inhibition efficiency/%,[25]) and exhibited qualitatively comparable behaviour to the case we presented here. Since subsequent predictive models trained on the thereby identified feature sets did not lead to relevant performance increase, we elected to only present inhibition efficiency/% results which are directly comparable to our previous study[10]. Additional information regarding this metric as well as results from the related feature selection process can be found in the Supplementary Notes as well as Supplementary Table 1.

Table 2. Identified feature sets.

FS_{60}	FS_{75}	FS_{15}
* P_VSA_MR_5	* P_VSA_MR_5	E1s
* LUMO	* LUMO	SHED_DL
* E1p	* E1p	MATS6e
CATS3D_02_AP	** HOMO	** HOMO
Mor04m	** E2s	** E2s

Best feature sets identified by RFE. FS_{60} denotes the set found using DS_{60}, and so forth. Features that occur in more than one winning set marked with single (overlap between FS_{60}/FS_{75}) and double (overlap between FS_{75}/FS_{15}) asterisks respectively.

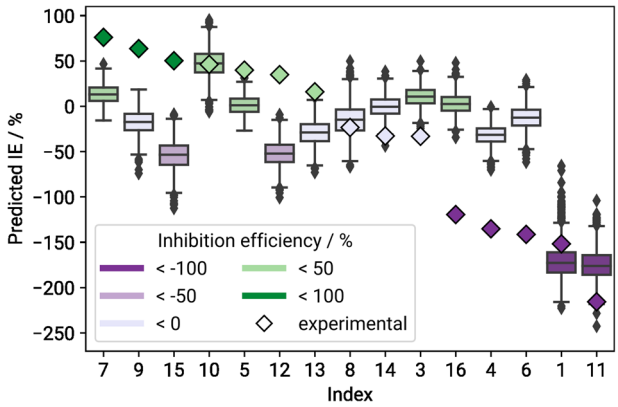


Fig. 2 Distribution of predictions across all cross-validation folds and random seeds per compound in the blind testing set, for neural networks trained on the original feature set FS_{60} and dataset DS_{60}. Boxes are coloured according to the compound's mean predicted IE values in %. Compounds are sorted by descending mean experimental IE values, which are depicted as coloured diamonds.

Generalisation ability of predictive models

One very important concern is the question of how well predictive models trained on the original data are able to generalise and capture the properties of completely unseen (i.e., blind testing) data. To this end, we repeatedly fitted a deep neural network on DS_{60}, using only inputs based on the associated winning feature set FS_{60}. The training data were split into the same 10 cross-validation folds that we used during the feature selection process, and on each fold the network was trained 100 times using varying random seeds. The blind testing dataset DS_{15} served as a completely unseen test set. Figure 2 shows the distribution of predicted inhibition efficiency values per compound in the blind testing set, aggregated over all cross-validation folds and random seeds. The detailed prediction means and standard deviations are provided in Supplementary Table 2.

Only about half of the compounds in DS_{15} get predicted correctly or within reasonable margins of error. The resulting root mean squared error (RMSE) for the blind testing set is fairly high at 73 percentage points (pp), cf. Table 3 for more statistics. We can see that the models have a tendency to underestimate inhibitors (i.e., compounds with IE > 0), but overestimate accelerators, as can be seen also in previous studies[21,26]. It is also notable that all but two prediction means lie within approximately ± 50 IE, which is where the majority of both the original as well as blind testing target values are situated. It is a common problem in machine learning that simply predicting the mean value of the target variable distribution might lead to a lower training loss than trying to find more complex dependencies. This behaviour can be

Table 3. Prediction statistics.

Statistic	FS_{60}/DS_{60}	FS_{60}/DS_{15}	FS_{60}/DS_{75}
RMSE	72.99	52.21	61.62
R^2	0.36	0.74	0.64
r	0.60	0.86	0.80
p	0.02	< 0.01	< 0.01

Root mean squared errors (RMSE), coefficients of determination (R^2), correlation coefficients (Pearson's *r*) and *p*-values of predictions on the blind testing dataset, per feature set. RMSE is given in percentage points (pp) w.r.t. the inhibition efficiency.

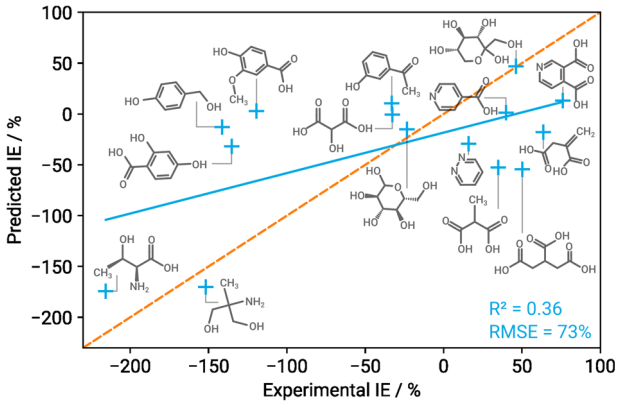


Fig. 3 Mean predicted inhibition efficiency values across all cross-validation folds and random seeds for compounds in the blind testing set, for neural networks trained using FS_{60}/DS_{60}. The solid blue line marks the resulting linear regression curve, the dashed orange line represents perfect fit.

indicative of overfitting or a suboptimal network architecture[27]. Figure 3 shows the average predicted over experimental IE, with the solid blue line representing the resulting linear regression curve, and the orange dashed line marking the perfect fit.

Overall we can conclude that the model trained on the original dataset, with features selected only for those data (denoted FS_{60}/DS_{60}), is able to predict the behaviour of completely unseen components only moderately well. This does not come as a huge surprise for two main reasons: Firstly, there is no overlap between FS_{60} and FS_{15}. This need not necessarily mean that FS_{60} is entirely unable to adequately capture the properties of compounds from DS_{15}, but it is an early indicator for results of reduced quality. Secondly, with only 60 samples in the original dataset we have to expect overfitting both for the feature selection process and especially the training of deep neural networks. The network architectures in Schiessler et al.[10] where chosen to vary as little as possible across a range of input feature counts, leading to overparameterised networks especially when working with very few features. With more fine-tuning of the network architecture and training hyperparameters, improved results might well be possible even on the blind dataset. However we can also make use of existing outliers to both gain important insights into the predictive domain of our models, as well as better understand the involved corrosion processes, or even identify yet unknown aspects of corrosion. In the following section we therefore include an extensive discussion of several components that obtained particularly conspicuous results.

Outliers

In Fig. 2 there are six compounds which are particularly salient, and which we consider to be strong outliers from the perspective

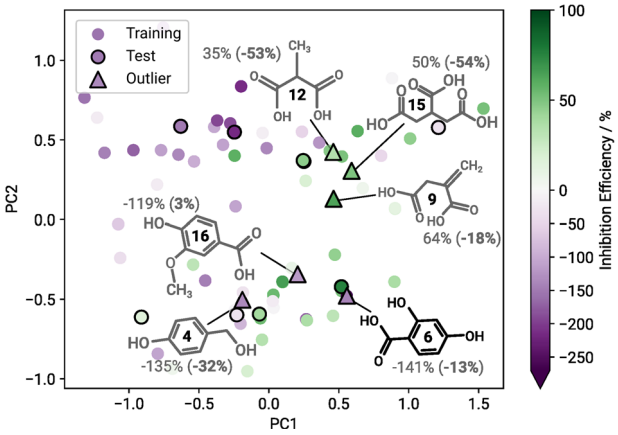


Fig. 4 Kernel principal component analysis of the molecular similarities for all 60 compounds of the original dataset (Training) and 15 blind testing chemicals (Test). Compounds identified as extreme outliers are marked accordingly (△) and illustrated along with their measured (**predicted**) inhibition efficiency. Predictions from FS_{60} / DS_{60} experiments.

of our deep learning models, cf. Fig. 4. These are compounds 9, 12, and 15, which are moderate to strong inhibitors but get qualitatively mispredicted as mild to strong accelerators, as well as compounds 4, 6 and 16, which are very strong accelerators but get predicted as only mild to moderate accelerators.

To better understand potential reasons why these compounds appear as outliers for the prediction models, deeper insights into their molecular structure shall be given. Analogously to Fig. 1, a structure-property landscape was generated for the total dataset of 75 compounds, where the compounds we consider to be outliers are marked accordingly (see Fig. 4). Analysing the resulting map, regions where compounds exhibit a similar corrosion inhibition efficiency indicate a structure-property relationship. Generally, it appears that corrosion accelerators are predominantly on the left side of the map and corrosion inhibitors on the right. Additionally, the structures are split into aliphatics (top side of the map) and aromatics (bottom side of the map).

2,4-Dihydroxybenzoic acid (compound 6) is located in a cluster predominantly populated by corrosion inhibitors, although experimentally it turns out to be a strong corrosion accelerator. It was still qualitatively correctly predicted as an accelerator. Compounds 6 is projected directly on top of 3,4-Dihydroxybenzoic acid, the strongest corrosion accelerator (-270% IE) of the original dataset. However the strongest corrosion inhibitor present in the blind testing set, 3,4-Pyridinedicarboxylic acid (compound 7), is located in the direct proximity as well. Apparently both corrosion inhibitors as well as accelerators contain mutual features in this region, rendering them similar in structure, even though they show different behaviours in the experiment. The trained models recognised a corrosion accelerator based on the selected features, but did not capture the subtle features that distinguish a strong from a weak accelerator, which is why the IE was overestimated. The overestimated IEs of 4-Hydroxybenzylalcohol (compound 4) and vanillic acid (compound 16) are situated in the same area of the map and can be explained accordingly. The structure-property relationship is not obvious in this region, as the compounds projected onto this area of the map exhibit structural features that are connected to varying corrosion inhibition efficiencies. Additionally, the experimental values of the three compounds 4, 6 and 16 lie at the lower edge of the target data distribution, further complicating accurate predictions. Adding more data points to this region, i.e., experimentally testing more compounds that exhibit similar structural features, is likely to improve the prediction performance for this domain.

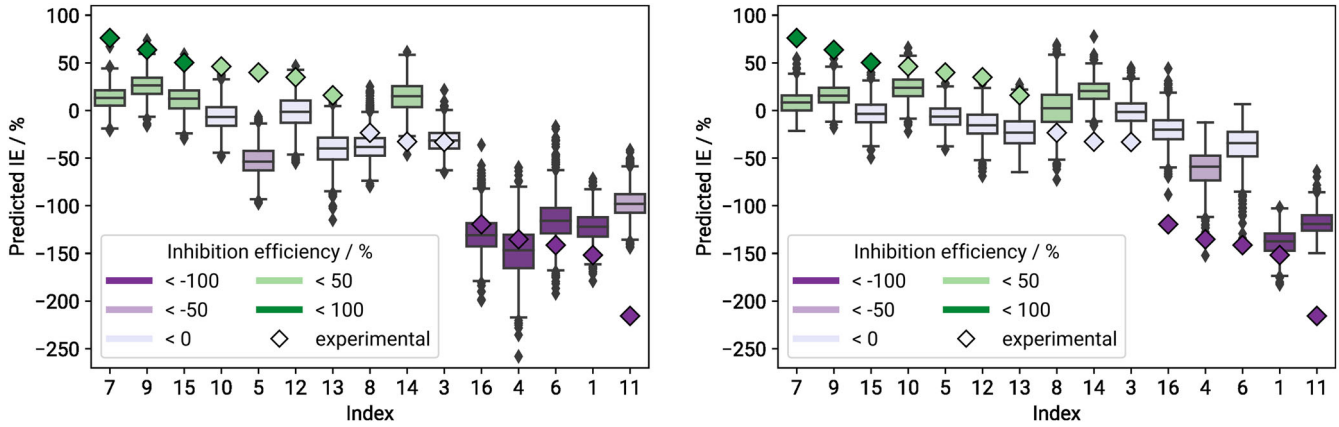


Fig. 5 Distribution of predictions across all cross-validation folds and random seeds per compound in the blind testing set, for neural networks trained using FS_{15}/DS_{60} (left) and FS_{75} / DS_{60} (right). Boxes are coloured according to the compound's mean predicted IE values in %. Compounds are sorted by descending mean experimental IE values, which are depicted as coloured diamonds.

Analysis of compounds 9, 12 and 15 shows that they were projected close to a region populated by weak corrosion inhibitors and accelerators. All of these compounds yield a moderate IE in the experiment and are mapped close to each other onto the structure-property landscape. The significant underestimations of the IEs probably stem from the absence of comparable corrosion inhibitors in this region. Furthermore, the selected features do not seem to capture the occurring structure-property relationship here accurately. However, future predictions for this region of the structure-property landscape are expected to improve with additional data.

Generalisation ability of the winning feature sets

In order to guarantee comparability to Schiessler et al.[10], we abstained from adjusting network architecture and training details in this work. Instead we examined the influence of using "better" feature sets towards improving the predictive quality and ability to generalise of our neural networks. In particular, we investigated whether predictive models that were trained on the original dataset DS_{60} could be improved if selected features were more suitable for the blind testing data, i.e., when training occurred in combination with FS_{15} or FS_{75}.

Clearly this approach is not applicable in practice without already having experimental values available for any data we wish to investigate, as those values are already needed during the feature selection process. Therefore the following results should not be seen as claims to the predictive capabilities of our already existing models. We can rather consider them as a lower bar on how well we are able to do given feature sets that really generalise well (recall that we still did not use the blind testing data during training of these neural networks).

We repeated the training process for the neural networks, using DS_{60} along with the same cross-validation folds as before as our training data, and again aggregating predictions on the blind testing set across all runs afterwards. The only difference was that FS_{15} and FS_{75} features were used as input instead. With this approach we hoped to improved predictive quality on the blind testing compounds, as their most relevant properties now played a direct role in adjusting the deep learning weights. Distributions of predictions for the blind testing data generated by FS_{15} / DS_{60} and FS_{75} / DS_{60} models can be found in Fig. 5. Detailed prediction means are provided in Supplementary Table 2.

In fact, in both cases we saw a drastic increase in accuracy with much fewer outliers and reduced RMSE of 52 percentage points (pp) and 62pp for the models using FS_{15} and FS_{75} respectively compared to 73pp for the FS_{60} models, cf. Table 3. Especially in the case of using features devised from only the blind testing data, this RMSE is on par with what was presented in Schiessler et al.[10], but without ever seeing these data during the training process.

The hidden downside, however, is that FS_{15} / DS_{60} models capture the qualities of the original dataset much more inaccurately. The overall RMSE for predictions on both the blind testing set and validation splits for this case is the highest of all three at 80pp, opposed to 67pp for both the FS_{60} / DS_{60} and FS_{75} / DS_{60} models.

Scalability

In order to further validate our approach we repeated the training process with cross-validation splits drawn from the combined dataset DS_{75} (the same that were used to determine FS_{75}). In this setup, there are no more blind testing data as they were incorporated into the combined dataset, thus we only report results aggregated from the respective validation sets per fold. At 64pp, the RMSE of the FS_{75} / DS_{75} models is on par with the mean RMSE of 63pp reported in Schiessler et al.[10], demonstrating that previous results can be replicated with different training sets and were not a consequence of for example overfitting.

From a machine learning perspective, a 25% increase of the dataset is not huge, and most likely the properties of the original data will still dominate overall results. From an experimental point of view, however, a great amount of time and effort went into performing the required analyses and already slight improvements in predicting the inhibition efficiency of organic compounds go a long way. At any rate we were able to increase the domain of applicability of our predictive models by virtue of the combined dataset, confirming the scalability of our method.

Discussion

In this work we investigated how well the predictive model that performed best in our previous study[10] holds up under blind testing. To this end, 15 previously unused compounds were randomly selected using the ExChem Routine[21] and their inhibition efficiencies w.r.t. the magnesium alloy ZE41 were experimentally determined using the setup presented by Lamaka et al.[24], forming the blind testing dataset DS_{15}.

Feature selection based on RFE suggested that the five features determined via the original dataset DS_{60} might not be able to generalise very well as there was no overlap between the winning feature sets for DS_{60} and DS_{15}. However, when regarding both the original and blind testing data in the form of a combined dataset DS_{75}, winning features were a 3:2 mixture from the winners of both individual sets, indicating that the feature selection process is indeed robust and scalable when further information is added. It is

notable that the DFT-derived descriptor HOMO came up in the runner-up second best feature set for the original data, and was included in winning sets for both the blind testing and combined dataset compositions, and in general seems to contain important information w.r.t. the inhibition efficiency properties of magnesium dissolution modulators.

Predictive modelling using deep neural networks trained on the original dataset and feature set confirmed that the originally selected descriptors showed only moderate success in correctly identifying the IE of the blind testing compounds. Training the networks on the newly identified feature sets managed to drastically improve the predictive quality even though the blind testing data themselves were only used during the feature selection step but never included in the training process. In summary we conclude that the identified feature sets are not yet able to thoroughly cover large parts of chemical space of potential additive components and need to be updated on a regular basis as more and more experimental data become available. Yet, even when given knowledge only about a very limited amount of data, our method already has a demonstrated predictive power in estimating the inhibition efficiency of magnesium dissolution modulators. Scalability of the method was confirmed via training the neural networks on the combined dataset composition.

In general, the architecture of the neural networks appears to be overparameterised given that we only used a total of five input features for training. This occurred in order to ensure comparability to the original setup presented in our previous study[10]. We aim to address this in future works using automated neural architecture search such as developed by Schiessler et al.[28] which can be helpful in choosing a better suited network topology while limiting the risk of overfitting on the training data. One issue with regression type machine learning is that there is less punishment during the learning process when the model qualitatively mispredicts target values (e.g., a positive target value is predicted to be negative and vice versa). This can be mitigated using classification type models, however, once higher levels of granularity are desired (e.g., for discerning between moderate and strong accelerators or inhibitors), custom loss functions are required that take into account ordered classes.

Another goal for future extensions is to further explore outlier detection using other related approaches such as autoencoders which are restricted to the features used in the machine learning models, as was briefly touched upon in Schiessler et al.[10].

METHODS

Corrosion experiments

Since the dataset used to train the initial deep neural network in this study was extracted from the work of Lamaka et al.[24], the model validation by blind testing was carried out with the same experimental setup and under the same conditions. The inhibition efficiency (IE) of the compounds selected by the ExChem routine was calculated based on hydrogen evolution tests, in which the amount of evolved hydrogen due to the corrosion of magnesium is measured during immersion in a NaCl solution. 0.5 g of ZE41 Mg chips with the surface area of $490 \pm 15\,\mathrm{cm^2\,g^{-1}}$ from the same batch used in Lamaka et al.[24] were immersed in 0.5 wt.% NaCl solution without (reference solution) and with the untested compounds, respectively. The chemical composition of the ZE41 chips used for our experiments was identical to the work of Lamaka et al.[24] and is provided in Supplementary Table 3. The concentration of compounds was 0.05 M and the pH of solutions was adjusted to 7.0 ± 0.1 by adding NaOH. Compound 3 (3-Hydroxyacetophenone) was used at its saturation, which was measured as 0.03 M. Since compound 1 (2-Amino-2-methyl-1,3-propanediol) has alkaline properties, 0.05 M of this chemical was first dissolved in an HCl solution with a Cl^- concentration equivalent to that of a 0.5 % NaCl reference solution. This solution's pH was then adjusted to 7.0 ± 0.1 with NaOH, similar to the other solutions.

The hydrogen evolution measurements were repeated three times for each solution and the mean of the calculated IEs was used for the corresponding blind test data point. IE is defined as follows

$$\mathrm{IE} = \frac{V_{\mathrm{H_2}}^{0} - V_{\mathrm{H_2}}^{\mathrm{Inh}}}{V_{\mathrm{H_2}}^{0}} \cdot 100\%, \tag{1}$$

where $V_{\mathrm{H_2}}^{0}$ and $V_{\mathrm{H_2}}^{\mathrm{Inh}}$ are the volumes of H_2 evolved after 20 h of immersion in the reference NaCl solution and the NaCl solution containing the investigated chemical compound, respectively. More details on the hydrogen evolution tests are available in the original publication by Lamaka et al.[24].

Molecular similarity

We selected suitable blind test candidates by using the ExChem routine[21]. ExChem exploits molecular similarities to find structurally similar chemical structures in a given database with respect to a selected chemical compound of interest. We calculated the underlying molecular similarities using the Smooth Overlap of Atomic Positions (SOAP) kernel that represents a high-dimensional similarity representation for the considered molecular compounds[22,23]. For each given compound, a local environment is first defined in a spherical region of radius r_c around each atom and then built by a superposition of Gaussian functions with width ξ. The structural information around an atom that flows into the similarity measure is directly dependent on the size of r_c. Calculating the translationally and rotationally invariant overlap between two local environments results in the SOAP kernel. The kernel can be further raised to a power ζ for improved discrimination between small or large similarities. Averaging over all local atomic environments enables the calculation of a global similarity measure that contains the molecular similarities between all chemical structures in a given dataset.

Interpretation of the molecular similarities in high-dimensional space was facilitated by projection to a two-dimensional latent space and correlation with experimental data. Distant (dissimilar) or close (similar) structures in the high-dimensional space maintain their relationships in the low-dimensional space. By evaluating the relative positions of compounds with respect to the formation of clusters in the two-dimensional similarity landscape, we can reveal existing structure-property relationships.

Feature generation

First, the geometries of the 15 blind test molecules were optimized using the quantum chemical software package Turbomole 7.4.[29] at the TPSSh/def2SVP[30,31] level of density functional theory. The optimized structures were subsequently used as input for the cheminformatics software package alvaDesc 1.0[32] and combined with six properties (HOMO, LUMO, HOMO-LUMO gap (ΔE_{HL}) as well as C_p, C_v, μ calculated at 293 K) that are directly derived from the output of the performed DFT calculations to generate the same pool of 1260 molecular descriptors that have been used in our previous work[10].

Feature selection

In Schiessler et al.[10], features (i.e. molecular descriptors) were selected using both ANOVA[11–14] and recursive feature elimination (RFE[15]) with a random forest regressor[16–18] as the underlying selector, and the corrosion inhibition efficiency as the target variable. RFE is a feature selection method that fits a specified regression (or classification) model given the available training data, and then determines a number of features that least influence the predictive result. These features are excluded from

the available pool, and the whole process is repeated until only the desired number or features remain.

Both methods were used to identify the group of top three, five, as well as 63 (i.e. top 5%) features. In all cases, the experiments were performed 100 times with a fixed train-test split of the available dataset, and then the group was determined that got selected most often (i.e., the selection mode). Subsequent predictive models trained on the various feature groups identified the set of five features as determined by RFE to be the most relevant w.r.t. predictions of inhibition efficiency of the available dataset. A full 10-fold cross-validation analysis confirmed both the composition of the top performing group as well as its status as most relevant set of features for predictive modelling.

In this work, we investigated the robustness of previous feature selection results under expansion of the training data. The 15 compounds listed in Table 1 were added to the original dataset used in Schiessler et al.[10], resulting in a combined dataset of 75 compounds. The resulting dataset compositions were denoted by DS_{60}, DS_{15} and DS_{75}, respectively.

Since in Schiessler et al.[10] features selected by ANOVA and groups of three features found by RFE produced significantly worse results when used in predictive modelling, and the set of 63 features showed signs of having a high noise-to-signal ratio, we focused our robustness analysis on grouped selection using RFE for groups of five features only.

For each dataset composition, we repeated the steps described in Schiessler et al.[10], running RFE 100 times using various random seeds per cross-validation fold, in order to select the grouped top five features per setting. Cross-validating experiments, such as we are doing, means splitting available datasets into n equal parts, called the folds[33]. The same experiment is then run n times, where a different portion of the data is withheld each time and serves as validation set for this fold. In the end, predictive results on the validation sets are averaged across all folds. This method is especially relevant when working with small datasets, to reduce overfitting and to reduce the influence of potential outliers that may be contained withing the data[19,20].

On DS_{60}, the cross-validation folds reported in our previous study[10] we re-used. On the other dataset variations, separate folds were drawn. Note that DS_{15} on its own, consisting only of 15 samples, is too small to expect consistent results under cross-validation. The winning feature sets where the ones that got selected most often per cross-validation fold and random seed. We named these FS_{60}, FS_{75} and FS_{15}, respectively.

Predictive modelling

As before in Schiessler et al.[10], we used deep learning to evaluate the relevance of identified feature sets for predicting inhibition efficiency of magnesium modulators. Since we restricted the feature selection process to sets of five features, only the architecture for what were called 'small' networks in Schiessler et al.[10] was reused. Our deep learning networks thus consist of the following layers:

- An input layer accepting inputs from the selected five descriptors
- A Gaussian noise layer with hyperparameters $\mu = 0$ and $\sigma = 0.1$
- Three fully connected layers with 50, 20, and 10 units, respectively, all using `relu` activation
- An output layer with one unit and no activation

The Gaussian noise layer adds some randomness to each input during training, drawn from a normal distribution with mean μ and standard deviation σ, which helps to counter the risk of overfitting on the training data. This layer is only active during the training phase. The networks were trained for 25 epochs using an `Adam` optimiser with learning rate 0.01, and mean squared error (MSE) as the loss function.

As a preprocessing step, all data that get passed through the networks were scaled using min-max-scaling, with the target variable being scaled into the range [0, 1], and the input variables into the range $[-1, 1]$.

We applied the same cross-validation folds that were used during the feature selection process. On each fold and setting, the same architecture was trained 100 times using different random seeds. Detailed software specifications are included in the Supplementary Notes.

For statistical analyses such as calculating the root mean squared error (RMSE) of the models, predictions for each compound were first averaged across all cross-validation folds and random seeds. Note that for the scalability analysis presented in Section Scalability, the blind testing data were included in the cross-validation folds. Analyses in this section were therefore not performed specifically on the blind testing data, but on the validation set results from each cross-validation fold.

DATA AVAILABILITY

The data used for this study is available at Zenodo via https://doi.org/10.5281/zenodo.7780743.

CODE AVAILABILITY

The code used for this study is available at Zenodo via https://doi.org/10.5281/zenodo.7780743.

Received: 18 April 2023; Accepted: 8 August 2023;
Published online: 12 September 2023

REFERENCES

1. Feng, Y., Xiong, W., Zhang, J., Wang, R. & Wang, N. Electrochemical discharge performance of the Mg-Al-Pb-Ce-Y alloy as the anode for Mg-air batteries. *J. Mater. Chem. A* **4**, 8658–8668 (2016).
2. Vaghefinazari, B., Höche, D., Lamaka, S. V., Snihirova, D. & Zheludkevich, M. L. Tailoring the Mg-air primary battery performance using strong complexing agents as electrolyte additives. *J. Power Sources* **453**, 227880 (2020).
3. Deng, M. et al. High-energy and durable aqueous magnesium batteries: recent advances and perspectives. *Energy Stor. Mater.* **43**, 238–247 (2021).
4. Erlanson, D. A., Fesik, S. W., Hubbard, R. E., Jahnke, W. & Jhoti, H. Twenty years on: the impact of fragments on drug discovery. *Nat. Rev. Drug. Discov.* **15**, 605–619 (2016).
5. Fockaert, L. I. et al. ATR-FTIR in Kretschmann configuration integrated with electrochemical cell as in situ interfacial sensitive tool to study corrosion inhibitors for magnesium substrates. *Electrochim. Acta* **345**, 136166 (2020).
6. Wang, Y. et al. High-throughput calculations combining machine learning to investigate the corrosion properties of binary Mg alloys. *J. Magnesium Alloys* https://doi.org/10.1016/j.jma.2021.12.007 (2022).
7. Lu, Z. et al. Prediction of Mg alloy corrosion based on machine learning models. *Adv. Mater. Sci. Eng.* **2022**, 9597155 (2022).
8. Hughes, A. E. et al. Corrosion inhibition, inhibitor environments, and the role of machine learning. *Corros. Mater. Degrad.* **3**, 672–693 (2022).
9. Sutojo, T. et al. A machine learning approach for corrosion small datasets. *npj Mater. Degrad.* **7**, 18 (2023).
10. Schiessler, E. J. et al. Predicting the inhibition efficiencies of magnesium dissolution modulators using sparse machine learning models. *npj Comput. Mater.* **7**, 193 (2021).
11. Johnson, K. J. & Synovec, R. E. Pattern recognition of jet fuels: comprehensive GC × GC with ANOVA-based feature selection and principal component analysis. *Chemometr. Intell. Lab. Syst.* **60**, 225–237 (2002).
12. Kim, T. K. Understanding one-way ANOVA using conceptual figures. *Korean J. Anesthesiol.* **70**, 22–26 (2017).
13. Burgard, D. R. *Chemometrics: Chemical and Sensory Data* (CRC Press, 2018).
14. van der Vaart, A., Jonker, M. & Bijma, F. *An Introduction to Mathematical Statistics* (Amsterdam University Press, 2017).
15. Guyon, I., Weston, J., Barnhill, S. & Vapnik, V. Gene selection for cancer classification using support vector machines. *Mach. Learn.* **46**, 389–422 (2002).
16. Ho, T. K. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition* Vol. 1, 278–282 (IEEE, 1995).

17. Genuer, R., Poggi, J.-M. & Tuleau-Malot, C. Variable selection using random forests. *Pattern Recognit. Lett.* **31**, 2225–2236 (2010).
18. Chavent, M., Genuer, R. & Saracco, J. Combining clustering of variables and feature selection using random forests. *Commun. Stat. B: Simul. Comput.* **50**, 426–445 (2021).
19. Arlot, S. & Celisse, A. A survey of cross-validation procedures for model selection. *Stat. Surv.* **4**, 40–79 (2010).
20. Cawley, G. C. & Talbot, N. L. C. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.* **11**, 2079–2107 (2010).
21. Würger, T. et al. Exploring structure-property relationships in magnesium dissolution modulators. *npj Mater. Degrad.* **5**, 2 (2021).
22. Bartók, A. P., Kondor, R. & Csányi, G. On representing chemical environments. *Phys. Rev. B* **87**, 184115 (2013).
23. De, S., Bartók, A. P., Csányi, G. & Ceriotti, M. Comparing molecules and solids across structural and alchemical space. *Phys. Chem. Chem. Phys.* **18**, 13754–13769 (2016).
24. Lamaka, S. V. et al. Comprehensive screening of Mg corrosion inhibitors. *Corros. Sci.* **128**, 224–240 (2017).
25. Kokalj, A. et al. Simplistic correlations between molecular electronic properties and inhibition efficiencies: do they really exist? *Corros. Sci.* **179**, 108856 (2021).
26. Feiler, C. et al. In silico screening of modulators of magnesium dissolution. *Corros. Sci.* **163**, 108245 (2020).
27. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow* (O'Reilly Media, Inc., 2019).
28. Schiessler, E. J., Aydin, R. C., Linka, K. & Cyron, C. J. Neural network surgery: combining training with topology optimization. *Neural Netw.* **144**, 384–393 (2021).
29. Turbomole. V7.4. *A Development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2019 Since 2007*. https://www.scirp.org/(S(i43dyn45teexjx455qlt3d2q))/reference/ReferencesPapers.aspx?ReferenceID=768588 (2019).
30. Staroverov, V. N., Scuseria, G. E., Tao, J. & Perdew, J. P. Comparative assessment of a new nonempirical density functional: molecules and hydrogen-bonded complexes. *J. Chem. Phys.* **119**, 12129–12137 (2003).
31. Eichkorn, K., Weigend, F., Treutler, O. & Ahlrichs, R. Auxiliary basis sets for main row atoms and transition metals and their use to approximate Coulomb potentials. *Theor. Chem. Acc.* **97**, 119–124 (1997).
32. Mauri, A. alvaDesc: A tool to calculate and analyze molecular descriptors and fingerprints. *Methods Pharmacol. Toxicol.* **64**, 801–820 (2020).
33. Stone, M. Cross-validatory choice and assessment of statistical predictions. *J. R. Stat. Soc., B: Stat.* **36**, 111–147 (1974).

ACKNOWLEDGEMENTS

Funding by the Helmholtz Association is gratefully acknowledged. TW, BV, SL and CF gratefully acknowledge financial support from the Helmholtz Artificial Intelligence Cooperation Unit via the *AI*^{2} project (Projektnummer ZT-I-PF-5-102). The authors thank Thermo Fisher Scientific for providing a chemical database that was used to select additional compounds for model validation using the previously developed ExChem approach.

AUTHOR CONTRIBUTIONS

E.J.S., T.W., B.V., S.V.L., R.H.M., C.J.C., M.L.Z., C.F. and R.C.A. contributed to the conception and design of the study. C.F. and T.W. generated the molecular descriptor database and selected the compounds for experimental validation of the model. B.V. and S.V.L. conducted the validation experiments. E.J.S. did the theoretical analyses and wrote the supporting code. E.J.S., T.W., R.C.A and C.F. evaluated the quality of the presented models. E.J.S. and T.W. created the figures. E.J.S., T.W., C.F. and R.C.A. wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

FUNDING

Open Access funding enabled and organized by Projekt DEAL.

COMPETING INTERESTS

The authors declare no competing interest.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41529-023-00391-0.

Correspondence and requests for materials should be addressed to Christian Feiler or Roland C. Aydin.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Searching the Chemical Space for Effective Magnesium Dissolution Modulators: A Deep Learning Approach using Sparse Features

SUPPLEMENTARY MATERIAL

Elisabeth J. Schiessler[1], Tim Würger[2, 3], Bahram Vaghefinazari[2], Sviatlana V. Lamaka[2], Robert H. Meißner[2, 3], Christian J. Cyron[1, 4], Mikhail L. Zheludkevich[2, 5, 6], Christian Feiler[✉, 2, 6], and Roland C. Aydin[✉, 1]

[1]Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Geesthacht, Germany
[2]Institute of Surface Science, Helmholtz-Zentrum Hereon, Geesthacht, Germany
[3]Institute of Polymers and Composites, Hamburg University of Technology, Hamburg, Germany
[4]Institute for Continuum and Material Mechanics, Hamburg University of Technology, Hamburg, Germany
[5]Institute for Materials Science, Faculty of Engineering, Kiel University, Kiel, Germany
[6]Kiel Nano, Surface and Interface Science KiNSIS, Kiel University, Kiel, Germany

SUPPLEMENTARY NOTES

Feature Selection Robustness under Change of Target Metric

In Kokalj at al.[1], the authors discuss advantages and disadvantages of using inhibition efficiency as a performance target measure for magnesium corrosion modulators. They conclude that even though inhibition efficiency is normalised for inhibitors and has a strong relation to molecular surface coverage θ, its high non-linearity is a distinct disadvantage.

Inhibition efficiency η is defined in Kokalj et al.[1], Eq. (15), as

$$\eta = \frac{R_p^{\text{inh}} - R_p^{\text{blank}}}{R_p^{\text{inh}}}, \tag{1}$$

where R_p^{inh} denotes the "mean polarised resistance of the inhibited sample and R_p^{blank} is the mean value of the blank, non-inhibited sample" (*cf.* Kokalj et al.[1], Section 2.3.1).

The authors aim to mitigate some of the disadvantages of inhibition efficiency by introducing a modified measure via logarithmic transformation that they name inhibition power. While inhibition efficiency describes a percentage, inhibition power is given in decibel.

Inhibition power P_{inh} is defined in Kokalj et al.[1], Eq. (16), as

$$P_{\text{inh}} = 10 \cdot \log_{10}\left(\frac{R_p^{\text{inh}}}{R_p^{\text{blank}}}\right). \tag{2}$$

From a re-ordering of equation (1) such that

$$\eta = \frac{R_p^{\text{inh}} - R_p^{\text{blank}}}{R_p^{\text{inh}}} \;\Leftrightarrow R_p^{\text{blank}} = R_p^{\text{inh}}(1-\eta)$$

the argument of the logarithm in equation (2) becomes

$$\frac{R_p^{\text{inh}}}{R_p^{\text{blank}}} = \frac{R_p^{\text{inh}}}{R_p^{\text{inh}}(1-\eta)} = \frac{1}{1-\eta}.$$

We can thus express inhibition power P_{inh} in terms of inhibition efficiency η as

$$P_{\text{inh}} = 10 \cdot \log_{10}\left(\frac{1}{1-\eta}\right). \tag{3}$$

Conversely, inhibition efficiency η can be expressed in terms of inhibition power P_{inh} via

$$\eta = 1 - 10^{-P_{\text{inh}}/10}.$$

✉ email: christian.feiler@hereon.de, roland.aydin@hereon.de

We repeated the feature selection procedure using the three datasets DS_{60}, DS_{15} and DS_{75} as discussed in the main manuscript with the experimental inhibition efficiency target values converted into inhibition power as described in equation (3). The winning feature sets can be found in Supplementary Table 1.

As with inhibition efficiency as target metric, we see some overlap between the best feature set for the combined dataset, $FS_{75}P$, and the other two, but none between the winning sets for the original and blind sets, $FS_{60}P$ and $FS_{15}P$ respectively.

It is furthermore notable that the winning sets discerned from the original dataset for both target metrics, FS_{60} and $FS_{60}P$, overlap in four out of five features. The remaining two descriptors, CATS3D_02_AP for inhibition efficiency and H3m for inhibition power, came up in the corresponding second best sets of the respective other metric. This overlap of winners between metrics occurs for FS_{75} and $FS_{75}P$ in three out of five, and for FS_{15} and $FS_{15}P$ in two out of five features.

Again the DFT derived descriptors HOMO and LUMO, denoting the highest occupied and lowest unoccupied molecular orbitals, seem to play an important role in capturing corrosion inhibition properties of the regarded datasets.

Settings for the Computational Experiments

Our experiments were performed on a virtual machine running on a 24-core 2.1 GHz Intel Xenon Scalable Platinum 8160 processor, equipped with a Tesla V100 GPU card and 16GB of memory.

We used the following software packages and versions to run the associated code and analyses:

- `keras 2.4.0`
- `numpy 1.21.5`
- `pandas 1.4.2`
- `python 3.8.2`
- `scikit-learn 1.1.0`
- `scipy 1.4.1`
- `tensorflow 2.3.0`

SUPPLEMENTARY TABLES

Feature Selection using the Inhibition Power Target Metric

Supplementary Table 1 contains the results of the feature selection process when performed using inhibition power as its target metric. Results are discussed above in the supplementary notes.

Supplementary Table 1: Top 5 feature sets identified by RFE when using inhibition power as target metric. $FS_{60}P$ denotes the set found using DS_{60}, and so forth. Features that occur in more than one winning set are highlighted in grey scale.

$FS_{60}P$	$FS_{75}P$	$FS_{15}P$
P_VSA_MR_5	P_VSA_MR_5	E1s
LUMO	LUMO	P_VSA_LogP_3
E1p	HOMO	HOMO
H3m	Mor30p	Mor30p
Mor04m	Mor14s	TDB05s

Detailed Predictive Results

Supplementary Table 2 contains the experimental inhibition efficiency values as well as mean predictions on the blind set per compound and feature set configuration. The means are aggregated over all cross-validation folds and random seeds. All models were trained on the original dataset DS_{60}.

Predictions from the FS_{75} / DS_{75} experiments (cf. the section on Scalability) are not included in Supplementary Table 2 as the blind set was integrated into the combined dataset and resulting cross-validation folds used during training, and thus no longer exists in the sense of a separate set.

Chemical Composition of ZE41

The elemental composition of the ZE41 used in this study is provided in Supplementary Table 3. The values were taken from Lamaka et al.[2]

SUPPLEMENTARY REFERENCES

[1] Kokalj, A. *et al.* Simplistic correlations between molecular electronic properties and inhibition efficiencies: Do they really exist? *Corros. Sci.* **179**, 108856 (2021).

[2] Lamaka, S. V. *et al.* Comprehensive screening of Mg corrosion inhibitors. *Corros. Sci.* **128**, 224–240 (2017).

Supplementary Table 2: Experimental and mean predicted inhibition efficiency values in % and standard deviations per feature set/dataset combination, averaged over all runs and cross-validation folds.

Compound		Experimental	FS_{60} / DS_{60}	FS_{15} / DS_{60}	FS_{75} / DS_{60}
1	2-Amino-2-methyl-13-propanediol	−152 ± 6	-170 ± 21	-122 ± 16	-138 ± 14
3	3-Hydroxyacetophenon	−33 ± 4	10 ± 11	-31 ± 13	-1 ± 14
4	4-Hydroxybenzylalcohol	−135 ± 3	-32 ± 11	-148 ± 27	-61 ± 20
5	4-Pyridinecarboxylic acid	40 ± 2	1 ± 10	-53 ± 15	-6 ± 12
6	2,4-Dihydroxybenzoic acid	−141 ± 2	-13 ± 14	-114 ± 24	-36 ± 21
7	3,4-Pyridinedicarboxylic acid	76 ± 1	13 ± 11	13 ± 12	8 ± 12
8	D-Glucose	−23 ± 2	-15 ± 18	-37 ± 17	3 ± 22
9	Itaconic acid	64 ± 3	-18 ± 13	26 ± 13	16 ± 11
10	L-Sorbose	46 ± 2	47 ± 17	-6 ± 15	24 ± 13
11	L-Threonine	−216 ± 10	-174 ± 17	-97 ± 16	-118 ± 12
12	Methylmalonic acid	35 ± 3	-53 ± 14	-2 ± 17	-15 ± 15
13	Pyridazine	16 ± 1	-29 ± 13	-40 ± 18	-23 ± 16
14	Tartronic acid	−33 ± 4	-1 ± 12	14 ± 17	20 ± 12
15	Tricarballylic acid	50 ± 17	-54 ± 16	12 ± 14	-3 ± 13
16	Vanillic acid	−119 ± 13	3 ± 11	-129 ± 22	-20 ± 16

Supplementary Table 3: Elemental composition of ZE41 as published by Lamaka et al.[2]. The values are given in ppm or in weight percent (wt.%) when indicated.

Element	amount	Element	amount
Ag	9.00 ± 2.00	Ni	7.00 ± 4.70
Al	144.00 ± 10.10	Pb	7.00 ± 4.00
Ca	27.00 ± 0.30	Pr	647.00 ± 36.00
Ce	0.65 ± 0.03 %	Si	2.00 ± 2.00
Cu	19.00 ± 0.50	Sn	49.00 ± 2.80
Fe	15.00 ± 8.30	Th	0.29 ± 0.003 %
La	0.42 ± 0.02 %	Ti	12.00 ± 1.30
Mg	93.75 ± 0.17 %	Y	50.00 ± 0.00
Mn	79.00 ± 2.20	Zn	4.20 ± 0.10 %
Nd	0.24 ± 0.01 %	Zr	0.30 ± 0.01 %

Appendix B

Manuscript in preparation for submission

B.1 Manuscript 4

ECToNAS: Evolutionary cross-topology neural architecture search

E. J. Schiessler, R. C. Aydin and C. J. Cyron

E.J.S., R.C.A. and C.J.C. contributed to the conception and design of the study. E.J.S. wrote the supporting code, ran the computational experiments, created the figures, and wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the version that is to be submitted.

Reprinted from E. J. Schiessler, R. C. Aydin, and C. J. Cyron. ECToNAS: Evolutionary Cross-Topology Neural Architecture Search. *manuscript in preparation for submission*, 2024. 10.48550/arXiv.2403.05123,

ECToNAS: Evolutionary Cross-Topology Neural Architecture Search

Elisabeth J. Schiessler*,✉,[1], Roland C. Aydin*,✉,[1,2], and Christian J. Cyron[1,2]

[1] *Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Strasse 1, 21502 Geesthacht, Germany*

[2] *Institute for Continuum and Material Mechanics, Hamburg University of Technology, Eissendorfer Strasse 42, 21073 Hamburg, Germany*

February 2024

We present ECToNAS, a cost-efficient evolutionary cross-topology neural architecture search algorithm that does not require any pre-trained meta controllers. Our framework is able to select suitable network architectures for different tasks and hyperparameter settings, independently performing cross-topology optimisation where required. It is a hybrid approach that fuses training and topology optimisation together into one lightweight, resource-friendly process. We demonstrate the validity and power of this approach with six standard data sets (CIFAR-10, CIFAR-100, EuroSAT, Fashion MNIST, MNIST, SVHN), showcasing the algorithm's ability to not only optimise the topology within an architectural type, but also to dynamically add and remove convolutional cells when and where required, thus crossing boundaries between different network types. This enables researchers without a background in machine learning to make use of appropriate model types and topologies and to apply machine learning methods in their domains, with a computationally cheap, easy-to-use cross-topology neural architecture search framework that fully encapsulates the topology optimisation within the training process.

Keywords: neural architecture search, evolutionary algorithm, topology crossing, structured pruning, singular value decomposition

1. Introduction

Deep learning as a tool to understand complex relationships within data sets and make predictions on unseen data has been around since the advent of machine learning. In recent years it has become more and more popular not only in a huge number of scientific fields but also for example in commercial applications, which in turn again attracts more people to the idea that making use of deep learning can be beneficial for them. Yet while it may be becoming easier to get started on deep learning, actually using it to achieve 'good' results can hinge on a variety of factors.

One of these crucial aspects is selecting an adequate network architecture, which also means deciding which types of architectural elements should be included. The optimal choice may be highly dependent on the specific task and data set, and even with expert deep learning knowledge finding said optimum can require hours and hours of computation time. Especially in applied sciences there are often limits to the available computational budget for deep learning. Additional difficulties may include an unclear task definition, a limited amount of training data, or data that is of mixed types (e.g. image data combined with additional measurements in medical applications).

Neural architecture search is a specialised branch of automatic machine learning (auto ML) that aims to mitigate some of these problems by limiting the required amount of human expert knowledge. It can be applied to independently find the ideal network architecture given the specified task and available data.

This work extends our previously published method called *the Surgeon* [1], which introduced a resource-friendly neural architecture search algorithm for multilayer perceptrons, also known as feed forward neural networks (FFNNs), that works especially well on restricted computing resources and limited training data. We combine it with network growth operations from Chen et al. [2] and state of the art, resource-friendly structured pruning techniques based on Liu et al. [3]. The result is ECToNAS, a cost-efficient evolutionary cross-topology neural architecture search algorithm that works as a stand alone method and in particular does not require pre-training any high level selector algorithms or meta controllers. It re-uses network weights between different candidates, and integrates training of the final architecture within the search process, thus reducing the hypothetically required amount of training time by around 80% when compared to re-training each candidate network from scratch.

We provide two different modes that can be applied as per user requirements. Standard ECToNAS focuses on maximising the specified target metric (such as validation accuracy), and manages to outperform the naive baseline throughout all our experiments. By setting a so-called greediness parameter it is possible to influence ECToNAS's scoring function such that more compressed network architectures might get favoured even at the cost of a small reduction in target

* *equal contribution*

✉ elisabeth.schiessler@hereon.de, roland.aydin@hereon.de

metric. With this mode we are able to shrink network parameter counts by up to 90% while losing only around 5-7 percentage points (or less) in target metric results compared to the greedy version.

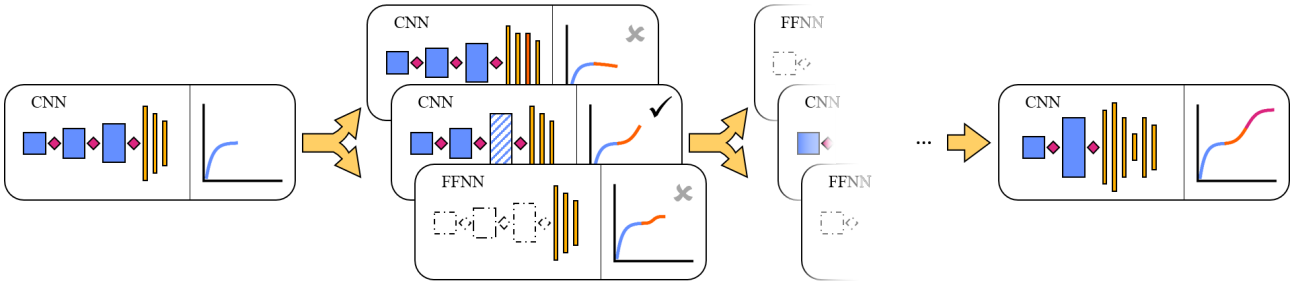


Figure 1: Graphical representation of the ECToNAS algorithm

ECToNAS is able to select suitable network architectures for different tasks and hyperparameter settings, and independently performs cross topology optimisation where required. By this we mean that starting from one topology type, such as a convolutional neural network (CNN), the combined training and topology optimisation may result in a different type of network, such as a standard FFNN, and vice versa.

The underlying code can be accessed via https://github.com/ElisabethJS/ECToNAS.

In this work we showcase the possibilities of re-using network weights in conjunction with cross topology optimisation and an evolutionary algorithm, and present some primers for further studies.

2. Related Works

Early versions of neural architecture search (NAS) have been around since at least Tenorio and Lee [4]. The idea of automated NAS for deep learning has been revitalised with groundbreaking works by Zoph and Le [5] and Liu et al. [6], which have seen a huge number of extensions in various directions [7], often relying on weight sharing between different network candidates [8–14].

Evolutionary algorithms that continuously optimize and evolve network topologies seem to be a promising approach [15]. However while reproduction-style algorithms such as Xue et al. [16] might be quite successful in their use of ResNet [17] and DenseNet [18] blocks, they rely on re-training from scratch an exponential amount per generation (23 GPU Days for CIFAR-10). More resource-aware approaches rely on incorporating pruning into the training process [19] or departing from the traditional two-stage setup, where first the optimal architecture is identified and then it is retrained from scratch and evaluated [11].

Pruning techniques can be roughly categorised into unstructured or structured approaches [20]. Unstructured pruning regards each connection within a neural network individually and identifies those that may be set to zero, leaving the overall sequence of layers and their topological specifics intact. The majority of publications on pruning techniques focuses on unstructured pruning which can show great results by using methods such as learning rate rewinding [21] or knowledge extraction [22]. Many attempts are made at reducing the required amount of retraining [3;23;24], with some focussing on various low-cost approximations and single shot estimators [25–31].

Structured pruning on the other hand aims to identify topological elements that can be removed completely. While there are some structured pruning approaches that are applied post training [32;33], others introduce additional regularization terms that promote desired sparsification already during training [34;35]. As an intermediate between structured and unstructured pruning, Zhang et al. [36] uses filters to align network elements.

NAS for image classification traditionally almost exclusively focuses on achieved validation accuracy as the target metric for deciding between various available model structures. In recent years, loss based evaluation metrics have been successfully applied especially when comparing model proxies instead of fully trained models [25–27;37], with Ru et al. [38] concluding that loss might in general be favourable over accuracy for candidate evaluation. None of the above approaches directly consider the number of network parameters in their decision metric, which can lead to quite large and over parameterised final architectures [39]. Some works penalise network size through different strategies, such as including some form of efficiency term within their loss function [19].

A truly merged cross-architecture NAS system such as ECToNAS has hitherto not been proposed to the best of our knowledge.

3. Methods

We begin by explaining the underlying methods and techniques that ECToNAS is built upon. The steps for manipulating multilayer perceptrons which were introduced in *the Surgeon* [1] and are used without further refinement or modification are briefly summarised in Appendix A.1. A graphical representation of the ECToNAS algorithm is provided in Figure 1.

3.1. Genetic Neural Architecture Search

Genetic algorithms (also called evolutionary algorithms) are inspired by natural processes such as evolution and mutation. A population of individuals competes towards some predefined goal, and the fittest members are able to reproduce and pass on (some of) their defining elemental structures. ECToNAS is a genetic algorithm that cyclically generates child populations C_i of neural network architecture candidates based off a number of parent networks P_i using mutation operations. A high level code summary is provided in Algorithm 1.

Algorithm 1 ECToNAS

function RUN(starting topology t_0)
    pre-train t_0 (warm start)
    initialise parent generation $P_0 = \{t_0\}$
    while termination criteria not met **do**
        # Mutation phase
        for each individual p **in** parent generation P_i **do**
            create potential offspring o_p, reject degenerates
        offspring from all parents form child generation $C_i = \bigcup_{p \in P_i} o_p$

        # Competition phase 1
        determine best competitor per operation type

        # Competition phase 2
        determine best remaining children C_i^* from all types

        # Evolution phase
        winning children become new parent generation $P_{i+1} = C_i^*$
    perform final selection to determine optimised topology t_{final}
    return t_{final}

These mutation operations not only change the network architecture (by adding or reducing neurons, channels or even whole layers), but also manipulate network weights in such a way as to incur minimal possible change in input-output behaviour between a parent and its modified offspring. This means adjusting network weights to fit to new shapes while preserving the maximal amount of information possible. The child generation then runs through two bracket style competitions in order to select the fittest individuals, which in turn become the new parent generation. Training occurs during the competition phases. The algorithm terminates when the allowed computational budget is spent.

We split selecting the fittest candidates into two phases in order to increase fairness during the selection process. More drastic changes will usually come at the cost of a somewhat reduced accuracy score, for which the network candidates need some training time to recuperate. Furthermore, smaller networks are able to converge faster than larger ones. Thus at first we only compare children that have undergone the same type of mutation, which all should have roughly the same pace of convergence.

In phase 1 of the competitions, only children of the same mutation type (for example adding channels to a convolutional layer) compete against each other, producing one winner per mutation type. Each pool of competitors is divided into pairs of 2 candidates. These receive a few epochs of training, then the one with the lower fitness score is eliminated from the pool.

In phase 2, all remaining offspring compete against each other, producing n winners C_i^* that in turn form the new parent generation P_{i+1}. n is a hyperparameter that can be set by the user. Higher n potentially yields better results as more individuals are retained in the parent generation, thus allowing ECToNAS to search a larger population of candidates, which in turn also increases the computational cost of the algorithm.

Training phases are already integrated into the competition process, and generating offspring re-uses already learned network weights, thus greatly reducing the required amount of training. Compared to *the Surgeon* [1], ECToNAS requires less overall training time and can access a much larger search space. Next to fully connected and various adaptor layers, ECToNAS's search space also includes convolutional, pooling, as well as batch normalisation layers, thus also allowing different topologies. It is able to independently decide which architecture type is best suited for the provided training data and task, and can add or remove structural elements as required and even change the type of network topology altogether.

3.2. Available Network Operations

During the mutation phase of Algorithm 1, ECToNAS has a number of modification operations available to generate mutated offspring from any given parent network. By design, these modifications change the network architecture as well as relevant weights, such that the overall input-output behaviour changes as little as possible. Chen et al. [2] have shown that under piecewise linear activation functions (such as `relu`), adding units to dense layers, channels to convolutional layers or even whole convolutional or dense layers can be performed with zero change to the overall network output. In Schiessler et al. [1], we demonstrated how to use singular value decomposition (SVD) to remove units from dense layers,

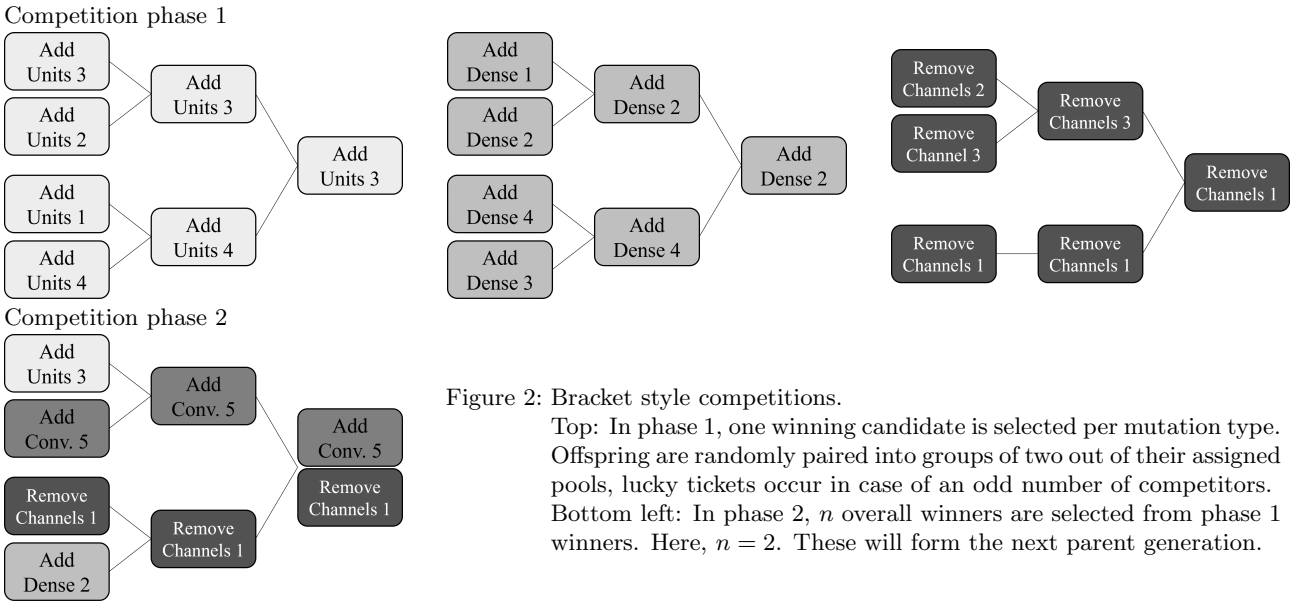


Figure 2: Bracket style competitions.
Top: In phase 1, one winning candidate is selected per mutation type. Offspring are randomly paired into groups of two out of their assigned pools, lucky tickets occur in case of an odd number of competitors.
Bottom left: In phase 2, n overall winners are selected from phase 1 winners. Here, $n = 2$. These will form the next parent generation.

or remove whole dense layers with as little information loss as possible. Summaries of these operations are included in Appendix A.1.

Unfortunately SVD cannot be easily extended towards convolutional layers. Instead we look at available pruning techniques. Since we do not wish to set individual connecting weights to zero, but actually remove all incoming and outgoing weights of for example a convolutional channel, we need to apply a structured pruning technique, see Figure 3. As discussed in Section 2, such methods often require expensive search algorithms, a large amount of re-training, or both; neither of which is feasible as each operation for ECToNAS which will get performed hundreds of times.

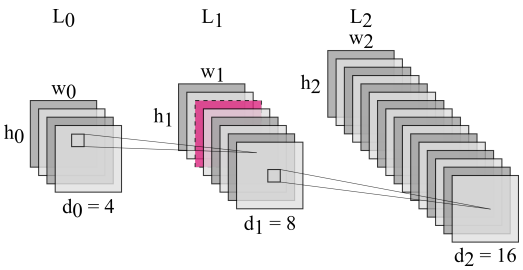


Figure 3: Example of a three layer CNN. Removing a channel from L_1 (coloured purple) affects all incoming and outgoing connections of that layer. Created using LeNail [40]

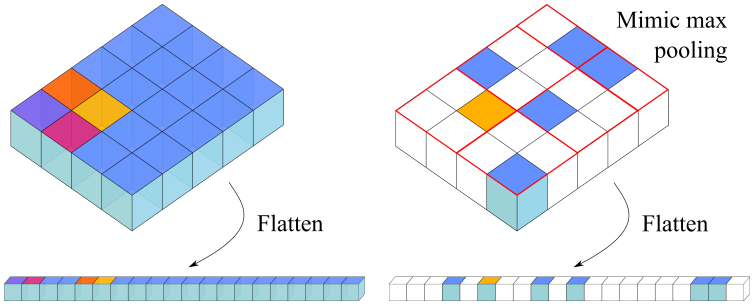


Figure 4: Left: Outgoing weights from the last convolutional layer are re-ordered by the flatten layer. Only one channel is depicted. Right: We mimic max pooling with stride 2, kernel size 2 on the outgoing weights of the last convolutional layer. White blocks are discarded.

Liu et al. [3] propose a pruning technique for convolutions based on the parameters of the following batch normalisation (BN) layer given by

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{1}$$

with input x, μ and δ representing the mean and standard deviation of the current mini-batch, scaling factor γ and offset factor β. There is one γ and β in a BN layer per channel in the previous convolutional layer. Liu et al. [3] argue that a small γ means that the corresponding channel has little influence on the layer's total output, and can thus be set to zero. The affected channel's β value is added to the bias term of the following layer. We adopt this idea and thus always pair convolutional with BN layers. When removing channels from a convolutional layer, we identify and cut away the ones corresponding to the smallest γ values, while adding the β term to the following layer's bias term. Similarly, removing a whole convolutional layer means cutting out the whole layer channel by channel. This type of operation greatly affects the overall input-output behaviour, and the weights of the newly generated child network are thus at best a warm start for further training, however we still find that this technique produces acceptable results and works as a trade off between computational cost and not having to completely re-train from scratch. In Liu et al. [3], the L1-penalty of γ is added to the loss function in order to promote pruning capabilities. Since this is not our main intent, we skip this term.

This leaves us with one more operation that we wish to include, namely pooling. Pooling layers work fundamentally different than all other layers we have discussed so far, as they have no weights of their own and cannot be initialised

to maintain overall input-output behaviour of the network. In terms of implementation, modifying a pooling layer does not affect the weights of successive convolutional layers (which are independent of respective input shapes), instead we need to look towards the first subsequent dense layer, which may be many steps down the line. We find that aggregating the input weights to the first dense layer as if we had applied the relevant pooling operation (that means using the same stride and aggregation function, see Figure 4) produces acceptable results and can again be seen as a warm start towards re-training. A more detailed explanation is included in Appendix A.2. Conversely, when removing a pooling layer we simply copy the incoming connections to the dense layer such that applying the pooling operation again would yield the original weights.

Adding convolutional layers without pooling drastically increases the overall network parameter count. Since we already always pair convolutional with BN layers, we instead define convolutional cells as fixed elements that may be added or removed only as a whole. Such convolutional cells are thus made up of the following combination (and order) of layers: convolution - pooling - batch normalisation - activation. Finally this gives us the pool of allowed network operations:

- Identity
- Add/remove dense layer
- Add/remove units to dense layer
- Add/remove convolutional cell
- Add/remove channels

These operations define the theoretical search space from which ECToNAS can identify winning topologies. In practice, the search space is further restricted by the provided starting topology and available computational budget.

3.3. Architecture Evaluation

Choosing a good balance between greedily optimising for validation accuracy or also rewarding smaller networks is not a trivial problem, as the optimal solution is not clearly defined and heavily dependent on available computational resources and task specifics. While validation accuracy has a clearly defined range of [0, 1] with 1 being the ultimate goal, there exist no hard limits nor a ubiquitously desirable optimum on network parameter count. Similarly when regarding loss instead of accuracy we not only have to change the goal towards decreasing the metric, but also keep in mind that achievable loss does not have a universal upper bound either.

In its fitness function, *the Surgeon* [1] weighs validation accuracy v against a function of accuracy gain Δv and parameter fraction Δp (both calculated with respect to a candidate's parent), amounting to $v + \exp(\Delta v)/\exp(\Delta p)$. This is based on Cai et al. [41]'s rationale that 'a 1% increase in validation accuracy should yield a higher gain if it occurs from 90% to 91% instead of 60% to 61%'. We find that the decision of how much focus to place on each aspect cannot be made globally and therefore make it dependent on a user choice parameter $\alpha \in [0, 1]$. ECToNAS thus has two modes: greedy ($\alpha = 1$) and balanced ($\alpha < 1$). The resulting fitness score s is given by

$$s = \begin{cases} v & \alpha = 1 \\ \alpha \Delta v - (1 - \alpha)\Delta p & \alpha < 1 \end{cases} \tag{2}$$

where again Δv and Δp are computed wrt. the candidate's parent. Note that competition phase 1 in Algorithm 1 is always evaluated greedily, and the balanced version is used in phase 2 if desired.

4. Experiments

We choose six different starting topologies for ECToNAS, three of which already contain one or more convolutional cells. The rest are simple feed forward neural networks of various sizes. Both the FFNNs and the topologies already containing one or more convolutional cells are further categorised by sizes ('small', 'medium' or 'large'), amounting to a total of 6 possible combinations. We perform 10 runs on each starting topology, with a computational budget of 1,000 total epochs per run for purely greedy mode ($\alpha = 1$) as well as in two non-greedy variants ($\alpha = 0.5$ and $\alpha = 0$). For our baseline we use standard training for 200 epochs performed on the same starting topologies. Experimental results are reported in detail for a fixed random seed and repeated twice more using different random seeds to confirm overall trends and reduce potential deterministic effects. Detailed descriptions of the starting topologies and further hyperparameters are included in Appendix C.2.

As an additional ablation study we deactivate the decision score and instead randomly select the winner in each competition bracket, which we call random mode.

Our experiments are performed across several standard image classification data sets: Cifar 10 & 100 [42], Eurosat [43;44], Fashion MNIST [45], MNIST [46], SVHN Netzer et al. [47]. Details on these data sets can be found in Appendix C.1. 10% of each training set are set aside as validation data, and a further 10% as test data in case no separate test split is available. Training and validation data are used by ECToNAS, the set aside test sets are only used for final evaluation purposes. ECToNAS keeps track of validation accuracy during training and uses these scores in the fitness function, as is discussed in section 3.3.

In Table 1 we compare the mean test accuracies over all runs per data set achieved by ECToNAS on greedy setting (E, $\alpha = 1$) with the baseline (B) and random mode (R). On the test set, ECToNAS outperforms the baseline in all cases, whereas the random mode scores similarly or slightly worse than the baseline with a small to moderate increase in average parameter count of the final architectures. Note that the parameter count of the baseline corresponds to that of the starting topologies, as these are not changed during standard neural network training.

Data set \ Mode	Test set accuracy [%]						Parameter count		
	E		B		R		E	B	R
CIFAR-10	54.2	± 3.4	46.3	± 4.7	47.1	± 3.5	100 K	77 K	74 K
CIFAR-100	20.6	± 3.1	16.0	± 1.9	14.8	± 3.8	97 K	78 K	80 K
EuroSAT	78.1	± 10.5	70.8	± 20.0	67.6	± 15.2	301 K	303 K	283 K
FashionMNIST	87.2	± 1.5	86.8	± 1.4	85.3	± 2.7	43 K	28 K	29 K
MNIST	98.3	± 1.6	97.6	± 2.2	96.0	± 3.4	43 K	28 K	29 K
SVHN	77.2	± 5.1	66.4	± 13.0	66.6	± 10.2	68 K	77 K	77 K
overall	69.3	± 26.0	64.0	± 28.7	62.9	± 27.6	109 K	99 K	95 K

Table 1: Performance on the test set and parameter count of final architecture of greedy ECToNAS (E, $\alpha = 1$) compared to unmodified baseline (B) and random mode (R) for various data sets. Data are aggregated over all different starting topologies.

By construction, each network candidate created by ECToNAS remembers the training that its parent has received, with as small memory losses as possible in case of drastic structural changes. Thus, the parent's count of received epochs of training are also included in the child's training count, such that the final optimised architecture produced by ECToNAS may have received between 100 to 160 epochs of training in total. When compared to re-training each candidate from scratch, as most NAS algorithms do, this means that ECToNAS can save up to around 80% of training time while still producing a fully trained final neural network. ECToNAS is limited by overall computational budget to be spread out over all generated candidates, thus we cannot directly control the number of epochs that the winner will receive. This number is influenced by a number of factors such as the size and structure of the initial topology and various hyperparameter settings which control for example the number of candidates generated and the maximum size of each parent generation. We see this difference when averaging over runs with different starting topologies for the same data set, where sudden jumps in otherwise relatively flat validation accuracy curves may occur due to some runs ending earlier than others. Figure 5 depicts average validation accuracies achieved by ECToNAS (greedy mode, $\alpha = 1$) compared to the native baseline and random mode. To make comparison fairer, we cut off the data for baseline and random mode for each starting topology after the same amount of epochs that ECToNAS on average spent on its winning topology. Apparent sudden jumps in accuracy levels (which do not recover within a few epochs) stem from averaging over runs with such varying cut-off points, such as for example the graph for the baseline trained on the EuroSAT data set in Figure 5, left hand side, middle row.

In Table 2 we compare results generated using ECToNAS with different greediness settings ($\alpha \in \{1, 0.5, 0\}$). In the greedy version ($\alpha = 1$), ECToNAS as expected beats the non-greedy versions wrt. final test accuracies by around 5-10 percentage points. It is noteworthy however that the non-greedy settings yield drastically compressed networks (achieving compression rates between 80% and 90%), and still in almost all cases manage to score at least as well as the baseline.

Data set \ α	Test set accuracy [%]						Parameter count		
	1		0.5		0		1	0.5	0
CIFAR-10	54.2	± 3.4	49.8	± 5.1	50.0	± 5.1	100 K	8 K	8 K
CIFAR-100	20.6	± 3.1	18.2	± 3.3	15.0	± 2.6	97 K	7 K	8 K
EuroSAT	78.1	± 10.5	71.3	± 7.1	70.7	± 8.7	301 K	36 K	17 K
Fashion MNIST	87.2	± 1.5	83.7	± 2.3	83.9	± 2.6	43 K	6 K	6 K
MNIST	98.3	± 1.6	94.7	± 3.2	94.8	± 3.2	43 K	6 K	6 K
SVHN	77.2	± 5.1	69.7	± 9.2	70.5	± 11.7	68 K	14 K	9 K
overall	69.3	± 26.0	64.6	± 25.5	64.1	± 26.8	109 K	13 K	9 K

Table 2: Performance on the test set and parameter count of final architecture compared over various ECToNAS greediness levels (α). Data are aggregated over all different starting topologies.

An example summarizing a single run of non-greedy ECToNAS ($\alpha = 0.5$) is presented in Figure 6. We can see ECToNAS's performance wrt. to both validation accuracy and network parameter count (left hand figure) as well as annotations describing which modifications were selected by the algorithm (right hand figure).

All our experiments are performed and reported for a fixed random seed (see Appendix C.2). In order to confirm

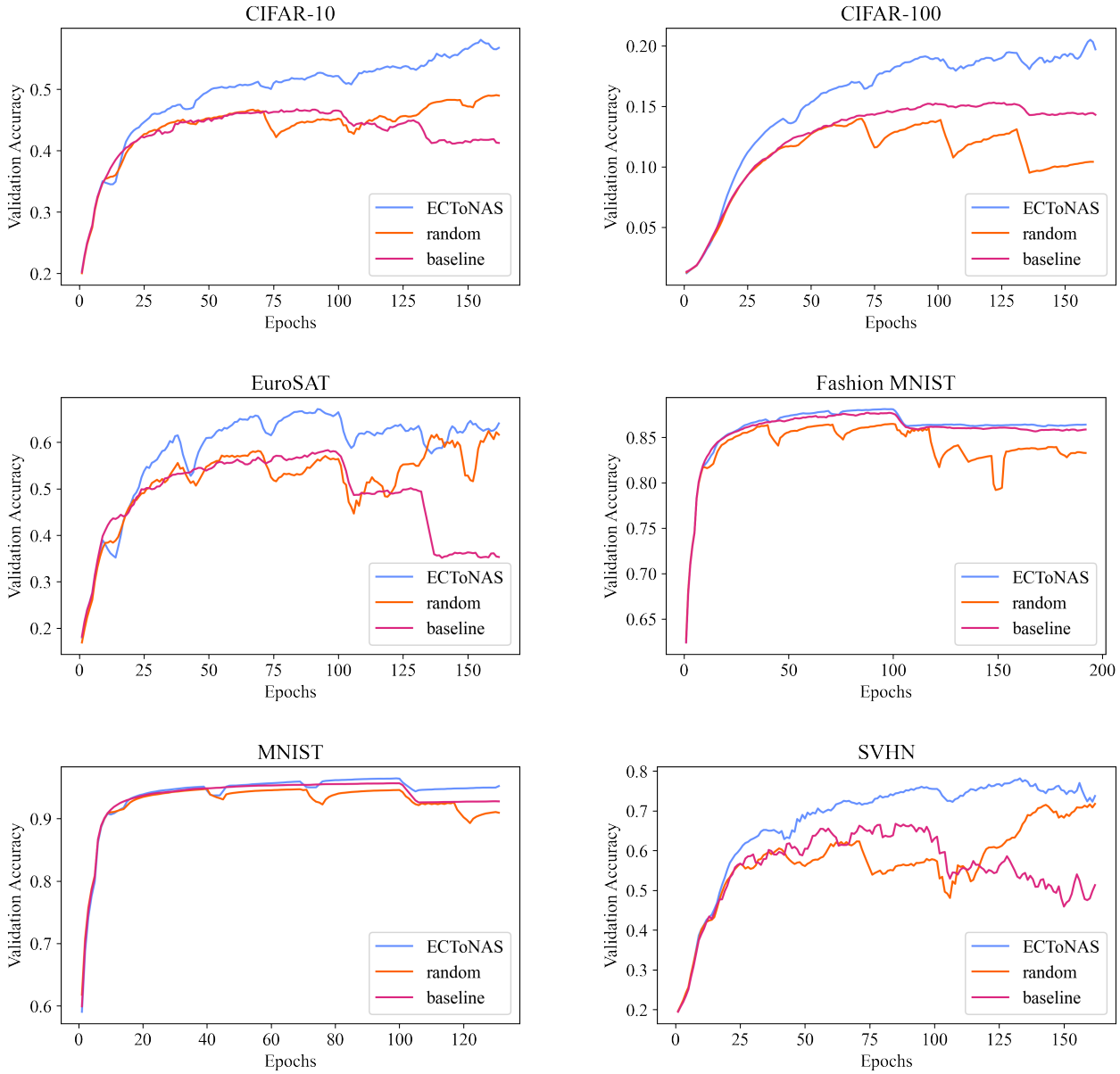


Figure 5: Comparison of accuracy on the validation set achieved by ECToNAS (greedy mode, $\alpha = 1$) with random mode and unmodified baseline topology across different data sets. Graphs are aggregated over all starting topology variations.

behaviour and limit the influence of unwanted deterministic effects we repeated all experiments twice more on different random seeds. These results fall in line with what has been presented.

The wall times per experimental run using the chosen data sets and starting topologies vary between 10 and 45 minutes, with most runs completing in around 20-30 minutes.

4.1. Convolutions on Tabular Data Set

Starting from a small initial topology and expanding layers or even introducing new topological elements as required is considered to be the most common use case for ECToNAS. However, it also has the ability to remove existing convolutional layers all together, leaving a fully connected feed forward neural network instead. This is handy when owing to unclear data type and objective function or simple lack of experience, a user might train a network with with one or several convolutional layers included that are not required or even harmful in order to achieve good results.

Since to the best of our knowledge there exist no well-known benchmark data sets for this use case, we artificially generate data that are structured like images (meaning tabular data will be formatted to look like $n \times m$ 'pixels'), but where there is no structural information contained. We base this off the well known adult data set also known as census data set,[48] see also Appendix C.1.

We one-hot encode any ordinal values, and drop any `nan`-columns, as well as the columns 'sex-Male' and the three

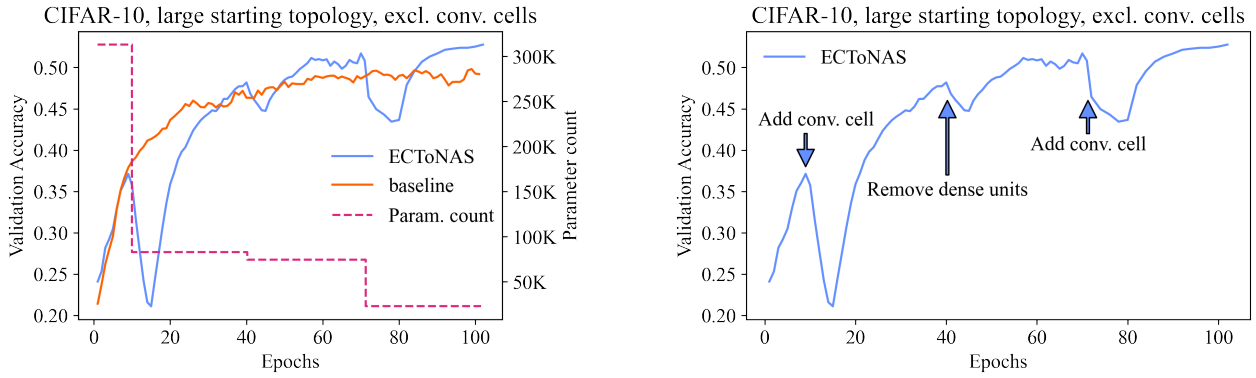


Figure 6: Example of a single run of non-greedy ECToNAS ($\alpha = 0.5$) on the CIFAR-10 data set starting off from a large topology without convolutional cells present. Left: Comparison with naive baseline and development of network parameter count during training. ECToNAS manages reach a higher validation accuracy level while simultaneously compressing the network to around 7% of its original size. Right: Modifications performed by ECToNAS are annotated.

native countries with the lowest count, in order to arrive at 100 columns. Numerical values are min-max-scaled into the range [0, 1]. Finally the column order is scrambled, and the data rearranged to mimic 10×10 'pixel' image data.

Using this artificially constructed data set, we run ECToNAS again on all six starting topologies that are described in Appendix C.2 and Table 6. We perform 25 runs with various greediness levels ($\alpha \in \{0, 0.5, 1\}$), as well as random and naive baseline mode. Thus 75 runs in total start off from either a CNN or a FFNN respectively for each mode. Again the greedy version of ECToNAS ($\alpha = 1$) manages to outperform all other modes with regard to test set accuracy. The final parameter count compared to the baseline goes down in case of the non-greedy versions, and up for greedy ECToNAS and random mode. Table 5 details these results and is included in the appendix.

For this experiment we are however more interested in the nature of the winning topology. Specifically we want to see how often topology crossing happens *away* from CNNs towards FFNNs. We expect to see few if any topology crossings towards FFNNs in non-greedy versions of ECToNAS, as the fitness function in these cases rewards smaller networks.

Starting topology	CNN			FFNN		
Final topology	CNN	FFNN	Crossing	CNN	FFNN	Crossing
Greedy ($\alpha = 1$)	70	5	6.7%	0	75	0.0%
Non-greedy ($\alpha = 0.5$)	75	0	0.0%	67	8	89.3%
Non-greedy ($\alpha = 0$)	75	0	0.0%	74	1	98.6%

Table 3: Counts of resulting topology types per starting topology type and greediness level, as well as percentage of runs that ended in topology crossing.

Indeed, in Table 3 we see this expected behaviour confirmed. Non-greedy versions of ECToNAS never cross towards a FFNN, as this would lead to increase in network size, and in most cases do cross towards CNN for the same reason. Greedy ECToNAS instead never crosses towards CNN, and in a few cases indeed removes all initially present convolutional cells (in addition to any that might have been added at run time) to end up with a standard FFNN. This can be seen as proof of concept that topology crossing indeed happens both ways when using ECToNAS.

5. Conclusion

We presented ECToNAS, a lightweight, computationally cheap evolutionary algorithm for cross-topology neural architecture search. One of its advantages over other neural architecture search algorithms is ECToNAS's ability to re-use network weights when generating new candidate networks, by manipulation of existing, pre-trained weights. To accomplish this we introduced several techniques for weight modification that aim to minimise the change in input-output behaviour caused by adding, removing, or changing structural elements of a neural networks' topology.

We showed that ECToNAS is able to supersede naive training, or reach comparable validation accuracy while being able to considerably compress the size of a given starting topology. The user is able to influence the desired outcome by tuning a greediness parameter, an can thus control how much focus should be placed on achievable validation scores vs. network parameter count. Ablation studies that deactivated its selection function demonstrated that ECToNAS is able to successfully identify those network candidates with the greatest potential.

ECToNAS tends to favour adding convolutional cells (a fixed sequence of the following layers: convolution, pooling, batch normalisation, activation) over removing them. The addition of the pooling layer shrinks network parameter count, which is rewarded in the fitness function, whereas the batch normalisation layer helps to stabilise the learning process.

We analysed and showed that topology crossing towards a FFNN, that is removal of all initially present convolutional cells, does happen as well under certain circumstances.

There are still some weaknesses inherent to this algorithm which were deemed to be out of scope for this manuscript but can be extended upon in future works. First and foremost, in its current state ECToNAS is restricted to sequential networks only. A possible starting point for how to get rid of this limitation may be found in Cai et al. [41]. There the authors show how to apply their Net2Net operations (which serve as basis for some methods we presented) to DenseNet, for example when using add operations. This limits achievable accuracy levels, and forces a somewhat outdated structure of any produced network architecture that is not comparable to the in general much more complex state-of-the-art models.

Secondly, ECToNAS is currently running on a very strict cyclical scheme. Mutation and selection phases are scheduled after a fixed amount of training, regardless of any internal states of the network candidates such as convergence rate etc. Introducing a dynamic learning rate may also serve to improve validation accuracy results. Early stopping could be used to further reduce computational requirements, but since the non-greedy version of ECToNAS at times selects candidates that perform worse wrt. their validation accuracy, it is somewhat unclear what criteria for early stopping could be applied that do not interrupt the algorithm too early.

Reaching state-of-the-art predictive accuracy levels is not everyone's goal however. Researchers without a strong background in machine learning often struggle to choose an appropriate architecture for their models, while potentially also having a limited availability of training data and/or computational resources. ECToNAS is ideally suited for these use cases since it can expand or shrink provided topologies as required and provides a fully trained, ready to use final neural network.

ECToNAS should thus be seen as a proof of concept that cross-topology neural architecture search is possible, and is intended to serve as a starting point for further research.

Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

A. Further Details on Network Operations

Here we will provide further details for network operations that either have already been published previously in Chen et al. [2] and Schiessler et al. [1], or would go into too much detail in the main manuscript body.

A.1. Fully Connected Layers

Chen et al. [2] introduce Net2DeeperNet and Net2WiderNet operations that can be used to add fully connected layers or widen existing layers.

When adding a new layer, its weight matrix is initialised as identity matrix and fully trainable. Under at least piecewise linear activation functions such as `relu`, this preserves the overall network output: Let ϕ be an activation function, then

$$\phi(v) = \phi(\mathcal{I}_d \phi(v)) \tag{3}$$

for all vectors v and identity operation $\mathcal{I}_d$, iff ϕ is at least piecewise linear.

Widening existing layers is done by copying and scaling some of the existing weights. Let fully connected layers L_i and L_{i+1} have weight matrices $W^{(i)} \in \mathbb{R}^{m \times n}$ and $W^{(i+1)} \in \mathbb{R}^{n \times p}$. The Net2WiderNet operation allows replacing L_i by a different layer L_i^* that has $q > n$ outgoing connections. For this, Chen et al. [2] define a random mapping function $g : \{1, 2, \ldots, q\} \to \{1, 2, \ldots, n\}$ such that

$$g(j) = \begin{cases} j & j \leq n \\ \text{random sample from } \{1, 2, \ldots, n\} & j > n \end{cases} \tag{4}$$

The weight matrices $W^{(i)}$ and $W^{(i+1)}$ are replaced by $U^{(i)}$ and $U^{(i+1)}$ with

$$U^{(i)}_{k,j} = W^{(i)}_{k,g(j)}, \; U^{(i+1)}_{j,h} = \frac{1}{|\{x|g(x) = g(j)|\}} W^{(i+1)}_{g(j),h} \tag{5}$$

The scaling factor is introduced to ensure that all units in the modified network have the same value as in the original network, even though weights may have been copied several times. Since we only applied linear operations, the same argument as above still holds true that Net2WiderNet preserves overall network output under at least piecewise linear activation.

As trimming information will in general change the overall input-output behaviour, *the Surgeon* [1] makes use of singular value decomposition (SVD) to remove neurons from a fully connected layer, or even whole layers: Given an arbitrary matrix $A \in \mathbb{R}^{m \times n}$ we can use SVD to find a representation

$$A = U \Sigma V^T \tag{6}$$

with orthogonal $U \in R^{m \times m}, V \in \mathbb{R}^{n \times n}$ and rectangular diagonal $\Sigma \in \mathbb{R}^{m \times n}$ with $k \leq \min(m, n)$ non-negative real entries σ_i along its diagonal. By convention, the σ_i (called singular values) are given in descending order of magnitude.

Reducing the rank of A to $r < k$ is done by setting $\sigma_i = 0,\ \forall i > r$ and dropping associated rows and columns of U and V.

Thus, $U\Sigma = \tilde{A} \in \mathbb{R}^{m \times r}$, with $\tilde{A}$ being the closest representation of A with rank r under Frobenius norm. $\tilde{A}$ can now serve as a new weight matrix for a dense layer with r units instead of n. In order to reconnect this layer to the following one, the subsequent weight matrix needs to be multiplied from the left with V^T.

In Schiessler et al. [1] we note that it is not possible to accommodate for effects of even linear activation functions using this method; thus the activation function is simply ignored and changes in input-output behaviour have to be mitigated with further training. Thus *the Surgeon* removes whole fully connected layers in a similar fashion by multiplying their weights onto the weight matrix of the subsequent layer, which preserves incoming and outgoing dimensions and ignores any present activation functions.

A.2. Pooling Layers

Pooling layers are structurally different from dense or convolutional layers in that they have no trainable weights of their own. As such, when we introduce a new pooling layer into an existing, pre-trained network, we cannot initialise the layer in such a way that it behaves as (close as possible to) an identity operation in terms of overall input-output behaviour. The second way in which the introduction of a pooling layer differs from other layer types is that this has structural consequences not only locally (meaning on the layer itself and its direct successor), but also further downstream up until the first dense layer, see Figure 7.

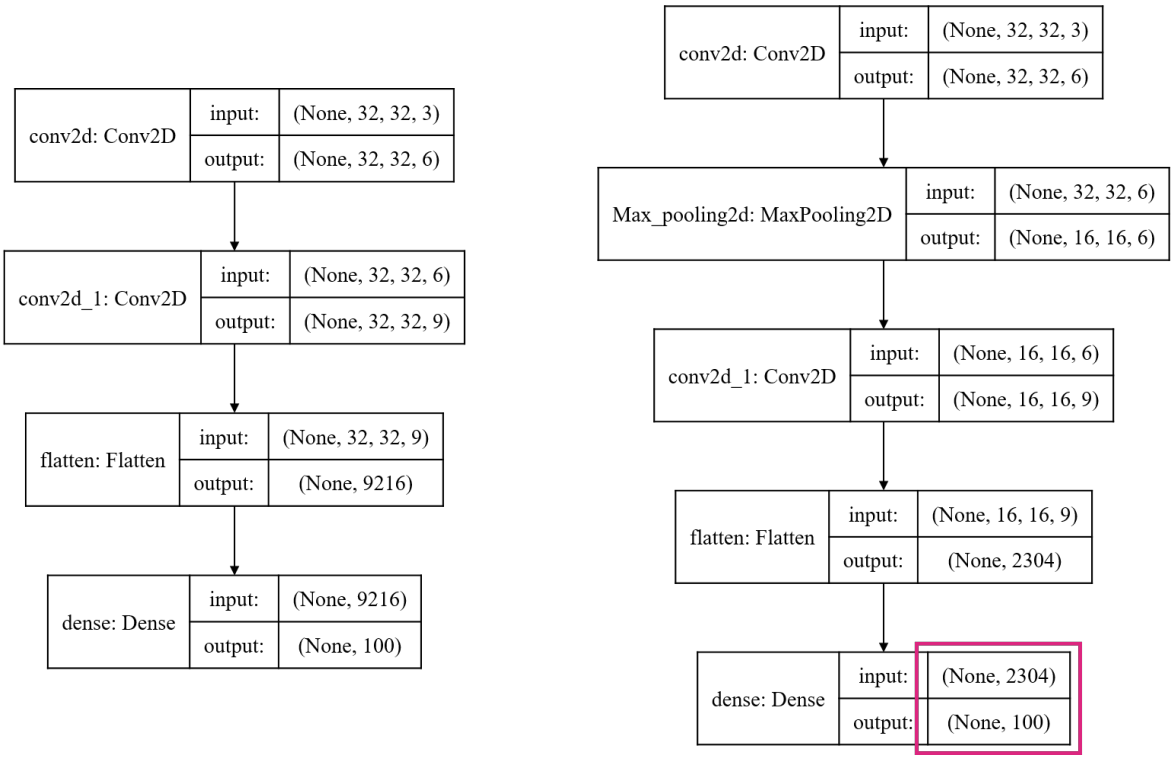


Figure 7: Introducing a pooling layer affects input and output shapes of all subsequent layers up to the first dense layer, where the weight matrix now has to accommodate for a different incoming dimension.

In particular, any convolutional or other layers in between the newly introduced pooling layer and the first dense layer also see the difference in input shapes, however their weights are not affected. Since we cannot retain (spatial) information that was lost through pooling, we opt to not concern ourselves with the affected convolutional layers and simply allow successive re-training to make up for lost accuracy. The same cannot be done with the first dense layer, which needs to have its weight matrix reduced to (w.l.o.g., depending on pooling layer settings and rounding effects) around $1/4^{th}$ of its original shape. A very simple approach would be to just randomly cut away superfluous input connections to the dense layer. Structured trials (discussed in A.2.1) have shown however that we can actually retain more information by mimicking the effect of pooling on the incoming connections to the flatten layer. Note that the flatten layer simply reshapes incoming information into vector format, such that the dense layer can parse it without any further altercations, see Figure 4 (left).

In the example provided in Figure 7, the flatten layer without pooling receives each training sample along a $(32 \times 32 \times 9)$ pixel weight matrix, or synonymously along 9 channels of (32×32) pixel weight matrices. We then introduce a max pooling layer with kernel size 2, stride 2 and zero-padding as required. The very top left 2×2 pixels of the first channel of each input sample have now been passed through max pooling, retaining only the single pixel which carries the most information. Which of the four pixels is selected will be different for each sample, and there is no way of knowing or

controlling that. We reason however that picking the outgoing weight of this 2×2 area with the largest absolute value has the greatest potential of retaining as much information as possible. The same of course applies to all other pixel groups in each sample.

Thus, when introducing a pooling layer, we apply the same aggregation function that is used for pooling (maximum or average) to each 2×2 weight block of the flatten layer input (using padding where required), and form a new weight matrix of the appropriate shape ($16 \times 16 \times 9$), which in turn is flattened to 2304 input connections.

Conversely, removal of pooling layers is done by repeating each incoming connection to the flatten layer the appropriate amount of times, and inserting them at the correct position. This works for both max and average pooling.

A.2.1. Pooling Layer Trials

To evaluate the soundness of the proposed pooling method we ran structured trials on a minimal example network consisting of two convolutional followed by two dense layers, which we trained for 25 epochs using the MNIST data set. This baseline network achieved an average validation accuracy of 98.6% over 100 different random seeds (resulting in different train-test-validation splits of the data set). We then inserted a pooling layer between the two convolutional layers and compared the loss in validation accuracy when adapting the weights using our proposed pooling method vs. randomly cutting away existing connections. The modified networks then each received one epoch of re-training, and validation accuracies where compared again. Results are summarised in Table 4.

Pooling method	Structured		Random	
Retraining	none	1 epoch	none	1 epoch
Average pooling	89.1%	96.0%	53.8%	95.6%
Max pooling	13.4%	29.7%	49.3%	95.0%

Table 4: Achieved validation accuracies of structured pooling trials using the MNIST data set on a minimal example network. Results are averaged over 100 runs using different train-test-validation splits. The baseline achieved an average validation accuracy of 98.6%.

From these trials we can see that the proposed method works really well in case of average pooling, but fails in case of max pooling, where it surprisingly achieves significantly worse results than just randomly cutting away connections. After one epoch of re-training, all but the structured max-pooling versions have regained a significant amount of the lost validation accuracy, whereas the max pooling layers are still quite far behind. We surmise that the proposed method shows promise especially when introducing average pooling layers where little retraining can be provided, and may be insufficient in combination with max pooling layers. Since we only tested on one data set and one example topology, there may well be cases where our method works also in the case of max pooling.

ECToNAS is able to independently select which type of pooling layer to apply when introducing new convolutional cells. Even with this type of pooling adaptation, we see cases where max pooling cells get chosen. Our proposed method of introducing pooling layers should thus be seen as a starting point for further investigations.

B. Additional Results

In this section we include further results that were omitted from the main body of the manuscript. Table 5 contains detailed results from experiments performed on the modified adult set. Note that the adult set has a very high class imbalance where the instances in both training as well as test set are split approximately 76%/24% across the two classes. This needs to be taken into account when comparing predictive results.

Mode	Greediness	Test set accuracy [%]	Parameter count
Baseline	-	84.4 ± 0.4	5.5K
ECToNAS	greedy ($\alpha = 1$)	84.8 ± 0.7	7.6K
	non-greedy ($\alpha = 0.5$)	83.6 ± 1.2	1.5K
	non-greedy ($\alpha = 0$)	83.5 ± 1.3	1.4K
Random	-	83.7 ± 1.2	6.5K

Table 5: Performance on the test set and parameter count of final architecture of the modified adult data set using different greediness settings for ECToNAS compared to random mode and naive baseline. Data are aggregated over all different starting topologies.

C. Implementation Details

In this section we provide details on data sets and our implementation of ECToNAS. The underlying code can be found at https://github.com/ElisabethJS/ECToNAS.

C.1. Data Sets

All image data sets were downloaded from the tensorflow data sets catalogue available at https://www.tensorflow.org/datasets/catalog/overview (last accessed on 2022-09-06). The tabular Adult data set was downloaded from the UCI Machine Learning repository[49] at http://archive.ics.uci.edu/ml/datasets/Adult (last accessed on 2022-09-06).

Adult[48]: Census income data set. Tabular data set containing 48,842 samples of categorical and numerical data across 14 attributes. Some missing values occur. Can be used to predict whether a person's income will be below or above USD 50K$/year. This data set has a heavy class imbalance.

Cifar 10 & Cifar 100[42]: Images from the Canadian Institute of Advanced Research. (32×32) px colour images that are evenly divided into 10 or 100 classes depending on the version. 50,000 training and 10,000 test images are available.

Eurosat[43;44]: Sentinel-2 satellite images. (64×64) px images. We use the RGB Version that has 3 colour channels and consists of 10 classes. 27,000 samples are available.

Fashion MNIST[45]: Data set of the fashion company Zalando's article images. (28×28) px grey-scale images evenly belonging to 10 categories. 60,000 training and 10,000 test images are available.

MNIST[46]: Modified National Institute of Standards and Technology data set of handwritten digits. (28×28) px grey-scale images evenly belonging to 10 classes that represent the digits 0-9. 60,000 training and 10,000 test images are available.

SVHN[47]: Images of house numbers from Google Street View. We use a cropped version of (32×32) px colour images that evenly fall into 10 classes representing the numbers 0-9. 73,257 training and 26,032 test images are available, as well as 531,131 extra training samples which are not used.

C.2. Hyperparameter Settings

We make use of a total of 6 different starting topologies which are summarised in Table 6. We differentiate between them by their size (with options 'small', 'medium' or 'large'), and by whether or not any convolutional cells are included. Convolutional cells consist of a CNN layer (with number of channels specified in Table 6), followed by an average pooling, batch normalisation and activation layer. We use `tensorflow`'s *padding='same'* option for both convolutional and pooling layers, and standard stride and kernel sizes, as well as `relu` activations. The dense layers also use `relu` activations, with respective unit count specified in Table 6.

Size	CNN cells included	conv. channels	dense units
small	true	3, 6, 9	10
	false	-	10
medium	true	3, 6	10, 10
	false	-	10, 10
large	true	3	100, 50, 10
	false	-	100, 50, 10

Table 6: Number of convolutional channels and dense units present in the various starting topologies used for ECToNAS experiments.

Note that the size descriptors relate to parameter count of the resulting neural network, which is why a topology that includes three convolutional cells (and thus a total of three pooling layers) is considered to be smaller than a topology that only contains one such cell. Each starting topology thus further consists of an input layer and a further dense output layer with `softmax` activation, with input shape and number of output channels tuned to the respective data set.

ECToNAS can be run with a number of different hyperparameters. Notable amongst these are the computational budget, greediness weight α and random seed, as well as indirect hyperparameters such as training and test set and optimizer, loss function and scoring metric.

For our experiments we chose the following settings:

- Computational budget: 1,000 epochs
- Greediness weight: 0, 0.5 and 1
- Random seed: 42 (extra experiments run on seeds 13 and 28)
- Optimizer: stochastic gradient descent, with learning rate 0.1
- Loss: sparse categorical cross entropy
- Scoring metric: accuracy

Further parameters can be set via code and are described in the documentation.

C.3. Python Environment Requirements

We used the following packages and versions to run ECToNAS. Important dependencies are marked with an asterisk.

- `python 3.8 *`
- `tensorflow 2.3.0 *`
- `keras 2.4.0 *`
- `numpy 1.21.5`
- `pandas 1.4.2`
- `re 2.2.1`

C.4. Hardware specifications

Our experiments were performed on a virtual machine running on a 24-core 2.1 GHz Intel Xeon Scalable Platinum 8160 processor, which is equipped with a Tesla V100 GPU card with 16 GB memory.

D. NAS Best Practice Checklist

When performing our experiments and writing up this manuscript we adhered to *the NAS Best Practices Checklist* (Version 1.0.1, available at https://www.automl.org/nas_checklist.pdf) as published by Lindauer and Hutter [50].

References

[1] E. J. Schiessler, R. C. Aydin, K. Linka, and C. J. Cyron. Neural network surgery: Combining training with topology optimization. *Neural Networks*, 144:384–393, 2021. 10.1016/j.neunet.2021.08.034.

[2] T. Chen, J. G. Ian, and J. Shlens. Net2Net: Accelerating Learning via Knowledge Transfer. In *International Conference on Learning Representations*, 2016. 10.48550/arXiv.1511.05641.

[3] R. Liu, et al. NFP: A No Fine-tuning Pruning Approach for Convolutional Neural Network Compression. In *2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 74–77. IEEE, 2020. 10.1109/ICAIBD49809.2020.9137429.

[4] M. Tenorio and W.-T. Lee. Self Organizing Neural Networks for the Identification Problem. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. https://proceedings.neurips.cc/paper_files/paper/1988/file/f2217062e9a397a1dca429e7d70bc6ca-Paper.pdf.

[5] B. Zoph and Q. Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, 2017. https://openreview.net/forum?id=r1Ue8Hcxg.

[6] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019. https://openreview.net/forum?id=S1eYHoC5FX.

[7] P. Ren, et al. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Computing Surveys*, 54(4), May 2021. 10.1145/3447582.

[8] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient Neural Architecture Search via Parameters Sharing. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, July 2018. https://proceedings.mlr.press/v80/pham18a.html.

[9] X. Dong and Y. Yang. Searching for a Robust Neural Architecture in Four GPU Hours. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019. 10.1109/CVPR.2019.00186.

[10] G. Bender, et al. Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 10.1109/CVPR42600.2020.01433.

[11] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann. Evaluating The Search Phase of Neural Architecture Search. In *International Conference on Learning Representations*, 2020. https://openreview.net/forum?id=H1loF2NFwr.

[12] Y. Xu, et al. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations*, 2020. https://openreview.net/forum?id=BJlS634tPr.

[13] L. Xie, et al. Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap. *ACM Computing Surveys*, 54(9), October 2021. 10.1145/3473330.

[14] L. Li, M. Khodak, N. Balcan, and A. Talwalkar. Geometry-Aware Gradient Algorithms for Neural Architecture Search. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=MuSYkd1hxRP.

[15] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan. A Survey on Evolutionary Construction of Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 25(5):894–912, 2021. 10.1109/TEVC.2021.3079985.

[16] Y. Xue, Y. Wang, J. Liang, and A. Slowik. A Self-Adaptive Mutation Neural Architecture Search Algorithm Based on Blocks. *IEEE Computational Intelligence Magazine*, 16(3):67–78, 2021. 10.1109/MCI.2021.3084435.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 10.1109/CVPR.2016.90.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 10.1109/CVPR.2017.243.

[19] C. Peng, Y. Li, R. Shang, and L. Jiao. ReCNAS: Resource-Constrained Neural Architecture Search Based on Differentiable Annealing and Dynamic Pruning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. 10.1109/TNNLS.2022.3192169.

[20] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the State of Neural Network Pruning? In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020. https://proceedings.mlsys.org/paper/2020/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf.

[21] D. H. Le, B.-S. Hua, D. H. Le, and B.-S. Hua. Network Pruning That Matters: A Case Study on Retraining Variants. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=Cb54AMqHQFP.

[22] E. M. Mirkes. Artificial Neural Network Pruning to Extract Knowledge. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. 10.1109/IJCNN48605.2020.9206861.

[23] D. Gurevin, et al. Enabling Retrain-free Deep Neural Network Pruning Using Surrogate Lagrangian Relaxation. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2497–2504. International Joint Conferences on Artificial Intelligence Organization, August 2021. 10.24963/ijcai.2021/344. Main Track.

[24] G. Tian, J. Chen, X. Zeng, and Y. Liu. Pruning by Training: A Novel Deep Neural Network Compression Framework for Image Processing. *IEEE Signal Processing Letters*, 28:344–348, 2021. 10.1109/LSP.2021.3054315.

[25] N. Lee, T. Ajanthan, and P. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. https://openreview.net/forum?id=B1VZqjAcYX.

[26] C. Wang, G. Zhang, and R. Grosse. Picking Winning Tickets Before Training by Preserving Gradient Flow. In *International Conference on Learning Representations*, 2020. https://openreview.net/forum?id=SkgsACVKPH.

[27] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6377–6389. Curran Associates, Inc., 2020. https://proceedings.neurips.cc/paper/2020/file/46a4378f835dc8040c8057beb6a2da52-Paper.pdf.

[28] S. Y. Arnob, R. Ohib, S. Plis, and D. Precup. Single-Shot Pruning for Offline Reinforcement Learning, 2021. 10.48550/ARXIV.2112.15579.

[29] T. Chen, et al. Only Train Once: A One-Shot Neural Network Training And Pruning Framework. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 19637–19651. Curran Associates, Inc., 2021. https://proceedings.neurips.cc/paper/2021/file/a376033f78e144f494bfc743c0be3330-Paper.pdf.

[30] J. N. Siems, A. Klein, C. Archambeau, and M. Mahsereci. Dynamic Pruning of a Neural Network via Gradient Signal-to-Noise Ratio. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. https://openreview.net/forum?id=34awaeWZgya.

[31] L. Miao, et al. Learning Pruning-Friendly Networks via Frank-Wolfe: One-Shot, Any-Sparsity, And No Retraining. In *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=O1DEtITim__.

[32] Z. Liu, et al. Learning Efficient Convolutional Networks through Network Slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017. 10.1109/ICCV.2017.298.

[33] Y. Idelbayev and M. A. Carreira-Perpiñán. Optimal Selection of Matrix Shape and Decomposition Scheme for Neural Network Compression. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3250–3254, 2021. 10.1109/ICASSP39728.2021.9414224.

[34] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. https://proceedings.neurips.cc/paper/2016/file/41bfd20a38bb1b0bec75acf0845530a7-Paper.pdf.

[35] Z.-S. Huang and C.-P. Lee. Training Structured Neural Networks Through Manifold Identification and Variance Reduction. In *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=mdUYT5QV0O.

[36] Y. Zhang, H. Wang, C. Qin, and Y. Fu. Learning Efficient Image Super-Resolution Networks via Structure-Regularized Pruning. In *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=AjGC97Aofee.

[37] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane. Zero-Cost Proxies for Lightweight NAS. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=0cmMMy8J5q.

[38] B. Ru, et al. Speedy Performance Estimation for Neural Architecture Search. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. https://openreview.net/forum?id=8V2hZW0d2aS.

[39] B. W. Larsen, S. Fort, N. Becker, and S. Ganguli. How many degrees of freedom do we need to train deep networks: a loss landscape perspective. In *International Conference on Learning Representations*, 2022. https://openreview.net/forum?id=ChMLTGRjFcU.

[40] A. LeNail. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*, 4(33):747, 2019. 10.21105/joss.00747.

[41] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient Architecture Search by Network Transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. 10.1609/aaai.v32i1.11709.

[42] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[43] P. Helber, B. Bischke, A. Dengel, and D. Borth. Introducing Eurosat: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 204–207, 2018. 10.1109/IGARSS.2018.8519248.

[44] P. Helber, B. Bischke, A. Dengel, and D. Borth. EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019. 10.1109/JSTARS.2019.2918242.

[45] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017. 10.48550/ARXIV.1708.07747.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 10.1109/5.726791.

[47] Y. Netzer, et al. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[48] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 202–207, 1996.

[49] D. Dua and C. Graff. The UCI Machine Learning Repository, 2017. http://archive.ics.uci.edu/.

[50] M. Lindauer and F. Hutter. Best Practices for Scientific Research on Neural Architecture Search. *Journal of Machine Learning Research*, 21(243):1–18, 2020. http://jmlr.org/papers/v21/20-056.html;https://www.automl.org/nas_checklist.pdf.

Appendix C

List of figures

Appendix D

List of tables

Appendix E

List of abbreviations

2D	2-dimensional
3D	3-dimensional
4D	4-dimensional
AE	Auto encoder
AGI	Artificial general intelligence
AI	Artificial intelligence
ANOVA	Analysis of variance
ANN	Artificial neural network
AutoML	Automated machine learning
BE	Backwards elimination
BIC	Bayesian information criterion
BN	Batch normalisation
CNN	Convolutional neural network
CV	Cross-validation
DARTS	Differentiable architecture search
DL	Deep learning
DFT	Density functional theory
DT	Decision Tree
ECToNAS	Evolutionary cross-topology neural architecture search
ELU	Exponential linear unit
FFNN	Feed forward neural network
GB	Gigabyte
GPT	Generative pre-trained transformer
GPU	Graphics processing unit
IE	Inhibition efficiency
LLM	Large language model
LSTM	Long short-term memory
LUMO	Lowest unoccupied molecular orbital
ML	Machine learning
MLP	Multi-layer perceptron
MSE	Mean squared error
MRI	Magnetic resonance imaging
NAS	Neural architecture search
NN	Neural network
PCA	Principal component analysis
pp	Percentage points
ReLU	Rectified linear unit
RF	Random forest
RFE	Recursive feature elimination
RMSE	Root mean squared error
RNN	Recurrent neural network
SFS	Sequential forward selection
SHAP	Shapley additive explanations
SVD	Singular value decomposition
ZE41	A specific magnesium alloy